Politechnika Wrocławska
Wydział Informatyki i Telekomunikacji

Kierunek: **Informatyka Algorytmiczna (INA)**

Specjalność: **-**

Praca Dyplomowa Inżynierska

Implementacja programu GRAPHPLAN do planowania akcji z wykorzystaniem programowania ograniczeń

Radosław Wojtczak

Opiekun pracy
dr Przemysław Kobylański

Słowa kluczowe:
Planowanie
Grafy
Programowanie ograniczeń
Sztuczna inteligencja

Wrocław 2022

Streszczenie

Obiektem badań poniższej pracy jest metodologia planowania o nazwie **"Graphplan"**, której esencją jest wykorzystanie struktury zwanej **grafem planującym** w trakcie ustalania optymalnego planu transformacji stanu początkowego w stan końcowy w ustalonej przestrzeni przy wykorzystaniu wcześniej zdefiniowanych operatorów.
Dodatkowym aspektem pracy jest użycie programowania ograniczeń w celu zwiększenia wydajności jak i zmniejszenia przestrzeni poszukiwań przez algorytm w trakcie generowania planu.
Ów praca składa się z formalnego opisu przytoczonego algorytmu, przedstawienia przykładów zastosowania, implementacji, której wynikiem jest graf, przedstawiający optymalny plan wykonywanych operacji oraz zmian, jakie dzięki nim zachodzą w świecie, omówienie opcjonalnych rozszerzeń, które w zależności od sytuacji mogą wpłynąć na efektywność algorytmu oraz przeprowadzonych testów, których zadaniem jest wskazanie mocnych, jak i słabych stron przedmiotu badań.
Grafy są generowane przy pomocy programu, który opiera się o moduł **graphviz**, dostępny w języku programowania **python**.

Abstract

Tutaj treść streszczenia po angielsku.

Spis treści

Spis rysunków

Spis tabel

Wstęp

Celem pracy jest zaimplementowanie algorytmu do planowania akcji o nazwie **GRAPHPLAN**, który po raz pierwszy został sformalizowany i opisany w pracy pod tytułem **"Fast Planning Through Planning Graph Analysis"**[2] przez Panów: Avrima L. Blum'a i Merricka L. Furst'a.
Praca składa się z szcześciu rodziałów.

W rozdziale pierwszym poruszono aspekty historyczne odnośnie planowania akcji przy użyciu komputerów oraz jaką rolę pełni w niej GRAPHPLAN, dokonano teoretycznego porównania algorytmu względem nowoczesnych metod z przytoczonej dziedziny informatyki.

W rodziale drugim poddano dogłębnej analzie implementowany algorytm- dokładnie opisano jego strukturę, warstwy, z których się składa oraz własności, które wyróżniają go na tle innych rozwiązań problemów związanych z planowaniem. W celu łatwiejszego przyswojenia mechanizmów stojących za GRAPHPLAN'em w trakcie opisu wprowadzono liczne proste przykłady wraz z grafikami wygenerowanymi przy pomocy narzędzi stworzonych na potrzeby ów pracy.

W rozdziale trzecim rozwinięto pojęcie programowania ograniczeń, wprowadzając formalną definicję, podstawowe słownictwo niezbędne do zrozumienia idei stojącej za tym sposobem programowania, omówiono benefity płynące z wykorzystania tego podejścia oraz przedstawiono obrazowo schemat funkcjonowania na podstawie prostych przykładów.

Rodział czwarty skupia się na szczegółach implementacyjnych: wybranych jezykach programowania oraz technologiach wykorzystywanych również w warstwach graficznych programu. Dokonano szczegółowego opisu interfejsu użytkownika oraz jego możliwości, połączeń między komponentami oraz ważniejszych funkcji stanowiących trzon pracy.

Rodział piąty przedstawia sposób instalacji oraz instrukcję obsługi programu, jak i instalowania wszystkich niezbędnych komponentów wykorzystywanych w pracy, w których skład wchodzą interpretery jak i kompilatory używanych języków programowania oraz wszystkie biblioteki i moduły.

Rodział szósty przedstawia przeprowadzone testy, które badają możliwości algorytmu w wcześniej spreparowanych środowiskach. W tej części została przeprowadzona analiza wydajnościowa algorytmu, weryfikacja wygenerowanych planów pod względem poprawności oraz porównanie otrzymanych wyników z innymi powszechnie wykorzystywanymi metodami planowania. Każdy z testów zawiera w sobie wniosek, w którym odbywa się zbiorcza ocena wszystkich wyżej wymienionych aspektów.

Końcowy rozdział stanowi zbiorcze podsumowanie pracy z komentarzem odnośnie potencjalnych rejonów, w których algorytm mógłby znaleźć swoje zastosowanie.

Rozdział 1

Wprowadzenie

1.1 Planowanie

Każdy człowiek codziennie wielokrotnie dokonuje procesu określonego mianem planowania, często nie zważając na to, jak skomplikowany proces wykonują. Skonstruowane przez ludzi plany mogą odnosić się do tak trywialnych zagadnień jak utworzenie listy zakupów, która wprost jest generowana przez braki w domowej lodówce oraz upodobania gastronomiczne kupującego, do bardziej abstrakcyjnych form jak *plan na życie*, czy *plan na wygranie meczu*. Przyglądając się planom oraz ich własnościom można wyodrębnić następujące trzy aspekty:

Definicja 1.1 ***Warunki początkowe*** - *stan świata przed zastosowaniem akcji. W dalszej części pracy również określane jako* ***stany początkowe****.*

- Każdy plan musi mieć jasno zadeklarowane warunki początkowe. Dzięki dokładnej wiedzy o świecie możliwym jest poprawne określenie akcji, przy pomocy których wprowadzane są modyfikacje obecnego stanu aż do otrzymania zadowalających rezultatów. Dla przykładu, firma musi wiedzieć ile oraz jakie palety przybędą na magazyn zanim rozpocznie planowanie rozkładu dostawy na magazynie.

Definicja 1.2 ***Akcja*** - *działanie zmieniające przedstawiony świat w ściśle określony sposób. W dalszej części pracy również określana jako* ***operator****.*

- Akcje pozwalają na modyfikację przedstawionego świata. Każda z akcji składa się z podmiotu, na który działa oraz czynności, która jest względem wskazanego podmiotu wykonywana. Przykładem dobrze określonej akcji może być przeniesienie klocka z jednego stolika na drugi- składa się ona z podmiotu w postaci klocka, oraz czynności w postaci przenoszenia, które możemy traktować w ogólniejszy sposób jako ruch. Czynności mogą różnić się od siebie w kwestii skomplikowania, najważniejszym jest, aby były określone poprawnie i aby były wykonalne w zdefiniowanym świecie.

Definicja 1.3 ***Cel*** - *Oczekiwany stan świata.*

- Kwintesencją każdego planu jest cel, który należy uzyskać. Zwyczajowo plany składają się z celów możliwych do osiągnięcia ze stanu początkowego przy pomocy zdefiniowany operacji, jednakże trzeba wziąc pod uwagę sytuację, w której niemożliwym jest uzyskanie wskazanego celu, szczególnie próbując automatyzować pojęcie planowania.

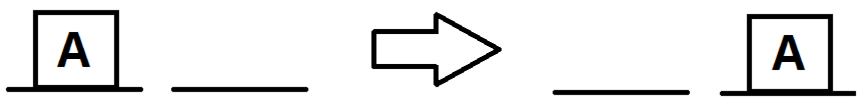
Przy pomocy powyższych definicji możliwym jest sformalizowanie pojęcia stojącego za słowem **plan**.

Definicja 1.4 ***Plan***- *lista akcji, której zastosowanie do stanu początkowego powoduje jego zmianę do stanu określonego w ramach cel.*

Oczywiście nie każdy plan musi być wykonalny. Zdarzają się również sytuacje, w których nie da się utworzyć planu dla zadanych warunków początkowych oraz celu. Z takimi przypadkami algorytm jest sobie w stanie sprawnie poradzić (o czym później w dalszej części pracy), jednakże istnieje inna grupa przypadków, która z persepktywy ów metodologii jest niemożliwa do realizacji. Mowa tu o sytuacjach, gdy któraś z akcji zmienia się w akcję **warunkową**.

Definicja 1.5 ***Akcja warunkowa*** *to typ akcji, która w zależności od bieżącego stanu świata produkuje inne wyniki*

Przykładem akcji warunkowej jest kopnięcie dmuchanej piłki. W zależności od wiejącego wiatru, które często zmienia się w sposób dynamiczny, przyłożenie tej samej siły do kopnięcia skutkuje różnym punktem końcowym trasy piłki. Ze względu na ograniczenia języka STRIPS plany w takim kontekście są niemożliwe do utworzenia, dlatego w dalszej części pracy wszystkie przedstawione sytuacje będą możliwe do całkowitego opisu przy pomocy nomenklatury STRIPS.



Rysunek 1.1: Przeniesienie klocka z jednej półki na drugą jako przykład sytuacji dla której istnieje możliwość utworzenia planu. Po lewej stronie od strzałki znajduje się stan początkowy, natomiast po prawej- oczekiwany cel. Naturalną akcją w przedstawionym świecie jest akcja *przenieś*, która zadany klocek przenosi z jednej półki na drugą.

Łatwo zauważyć na podstawie przykładu 1.1, iż można utworzyć wiele planów, które dla zadanego stanu początkowego osiągają wskazany cel. Dla powyższej sytuacji naturalnym planem jest przeniesienie klocka A z platformy po lewej na platformę po prawej, lecz nie jest to jedyna możliwość. Również satysfakcjonującym planem zgodnie z wprowadzoną wyżej definicją byłaby następująca sekwencja akcji:

1. Przenieś klocek A z lewej platformy na prawą

2. Przenieś klocek A z prawej platformy na lewą

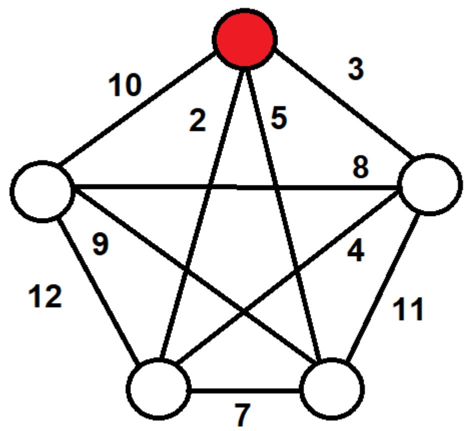
3. Przenieś klocek A z lewej platformy na prawą

Generowanie rekruencyjne nieskończonych planów poprzez bezużyteczne przestawienia ”w miejscu”, mimo tego, iż zawiera się w definicji 1.4, nie jest oczekiwanym efektem. Proces planowania odbywa się po to, by wykonać transformację świata przy jak najmniejszym nakładzie sił w jak najkrótszym czasie. Z tego względu wprowadzono pojęcie **planu optymalnego**.

Definicja 1.6 ***Plan optymalny****- Plan o minimalnej liczbie kroków satysfakcjonujący wskazany cel.*

W dalszej części pracy słowo **plan** najczęściej będzie utożsamiane z planem optymalnym.

Wprowadzenie powyższej definicji wiąże się z powstaniem bardzo ważnego pytania: *Jak utworzyć plan optymalny?*. Bazując na przykładach takich jak 1.1 złudną może być myśl, iż generowanie optymalnych planów jest rzeczą prostą. Niech kolejny przykład będzie tego dowodem.



Rysunek 1.2: Problem komiwojażera (Travelling Salesman Problem, TSP). Na rysunku pomocniczym liczby symbolizują wagi krawędzi. Węzeł oznaczony kolorem czerwonym odpowiada węzłowi startowemu, do którego należy wrócić. Długości krawędzi nie zachowują wskazanych przez wagi proporcji.

Problem komiwojażera jest popularnym zagadnieniem optymalizacyjnym, którego istotą jest wskazanie ścieżki o najmniejszej sumie wag krawędzi, która przechodzi dokładnie jeden raz przez każdy wierzchołek i wraca do wierzchołka startowego. Często problem wędrującego komiwojażera przedstawiany jest przy pomocy kuriera oraz domów, które musi odwiedzić (za wierzchołek startowy uważa się magazyn, w którym kurier rozpoczyna swoją pracę). Mimo faktu, iż w przykładzie 1.2 występuje jedynie 5 miejsc, w których musi zatrzymać się kurier, utworzenie optymalnego planu dla przedstawionej sytuacji jest nielada wyzwaniem. Z tego powodu ludzie postanowili skorzystać z potężnych mocy obliczeniowych komputerów przy generowaniu bardziej skomplikowanych planów.

1.2 Planowanie przy użyciu komputerów

1.2.1 STRIPS

W 1971 roku Panowie: Richard Fikes oraz Nils Nillson z Standford Research Institute (SRI International, jeden z najsłynniejszych na świecie ośrodków badawczych) zdecydowali się na przedstawienie światu nowego podejścia w dziedzinie planowania o nazwie **STRIPS** (**ST**anford **R**esearch **I**nstitute **P**roblem **S**olver)[6]. **STRIPS** rozwiązuje wskazany problem poprzez przeszukiwanie wszystkich stanów świata, aż do momentu gdy znajdzie taki, w którym wskazane cele są spełnione. Ważnym założeniem programu jest istnienie ciągu akcji, który gwarantuje otrzymanie celu. Zadanie to jest realizowane poprzez znalezienie sekwencji operatorów, która konwertuje wymodelowany stan początkowy, w model, w którym wszystkie zdefiniowane cele są spełnione. Defnicje operatorów, stanu początkowego oraz celu są niemalże identyczne jak 1.1, 1.2 i 1.3, z tym, że definicja Akcji została w naturalny sposób rozwinięta o ciąg przyczynowo-skutkowy. Zauważono, iż w skład każdej akcji poza samą czynnością wchodzą dwie składowe, nazywane środowiskowymi- warunki zajścia oraz efekty zajścia.

Definicja 1.7 ***Warunkiem****- zajścia akcji jest istnienie odpowiedniej konfiguracji świata, dzięki której akcja może zostać wykonana.*

Definicja 1.8 ***Efektem****- zajścia akcji są zmiany, które zaszły w przedstawionym świecie ze względu na jej wykonanie.*

Mówiąc kolokwialnie, każda akcja ma swoją przyczynę oraz swój skutek. **Przyczyną** akcji w przykładzie 1.1 jest znajdowanie się klocka na lewej platformie. Gdyby klocek A znajdował się na prawej platformie, wykonanie akcji przesunięcia klocka z platformy lewej na prawą nie mogłoby zostać wykonane, natomiastem **efektem** akcji jest przeniesienie klocka na prawą platformę. Łatwo zauważyć, iż brak innych obiektów na platformie prawej jest niezbędny, aby klocek mógł zostać tam przeniesiony. Kolejną naturalną obserwacją jest stwierdzenie, iż przeprowadzenie akcji dodaje nam nowe informacje o świecie w dwóch kontekstach:

- Dodającym- pojawienie się lub podtrzymanie danej składowej świata
- Usuwającym- pozbawienie świata danej składowej

Po wykonaniu czynności z przykładu 1.1 wyróżniamy trzy typy nowych informacji:

- Warunek- Obecnośc klocka na lewej platformie, prawa platforma jest pusta
- Efekt dodający- Obecność klocka na prawej platformie
- Efekt usuwający - Platforma lewa jest pusta

W rozważanym podejściu każda z czynności zdefiniowana jest przy pomocy wyżej wskazanych trzech składowych.

Takie zdefiniowane świata okazuje się wystarczające do rozwiązywania problemów pokroju rearanżacji obiektów czy nawigowania w ściśle zdefiniowanej przestrzeni, czego najlepszym przykładem, jest pierwszy robot do realizacji ów zadań- **Shakey**. Shakey był pierwszym robotem, który dzięki zainstalowanemu oprogramowaniu, posiadał umiejętność analizy własnego otoczenia. Dzięki zaimplementowanemu podejściu STRIPS (oczywiście z odpowiednimi dostosowaniami do sytuacji) był w stanie rozwiązywać problemy z zakresu wyznaczania drogi, czy planowania rozmieszczenia obiektów w pokojach. Ze względu na swoją innowacyjność i przełomowość często jest nazywany archetypem dzisiejszych autonomicznych samochód czy militarnych dronów.

Dzięki swojej roli w rozwoju planowania z użyciem komputerów, STRIPS został dodatkowo wyróżnionyod jego nazwy pochodzi języki opisu świata korzystający z trójki: stan początkowy, akcja oraz cel. Przez następne lata rozwiązania w obszarze planowania silnie bazowały na wprowadzonym w powyższej pracy opisie świata.

1.2.2 Rozwiązywanie ludzkich problemów

W 1972 roku Panowie Allen Newell i Herbet Simon opublikowali książkę pod nazwą **Human Problem Solving**, której tłumaczenie znajduje się w tytule opisywanej sekcji. Założenie odnośnie otrzymywanych przez algorytm rozwiązujący problem danych pozostało zgodne z opisem w sekcji 1.2.1, jednakże autorzy pracy zauważyli, iż utworzony w ten sposób obszar działań (*problem space*) może osiągać niebotyczne rozmiary. Chcąc odpowiedzieć na pytanie, w jaki sposób ludzie wybierają odpowiednie operacje w danej sytuacji, zauważyli, iż ludzie często wykorzystują *skróty*, które wynikają bezpośrednio z intuicji. Człowiek ze względu na swoją naturę będzie dążył do zużycia jak najmniejszych pokładów energii w celu osiągnięcia wyznaczonego celu. Na tej podstawie rozpoczęło się formowanie technik rozwiązywania problemów zwanych **heurystykami**. Początkowe heurtystki ograniczały się do zastosowania pewnych ograniczeń jeśli chodzi o wyznaczanie zbioru operatorów, na przykład poprzez unikanie powtórzeń (*repeat-state avoidance*) oraz unikanie powrotów do stanó już znanych (*backup avoidance*). Na przestrzeni następnych lat pojęcie heurystyki rozwinie się diametralnie, jednakże zachowa swoją pierwotną definicję szukania rozwiązań na skróty lub kolokwialnie mówiąc *na ludzkie oko*.

1.2.3 Liniowy i częściowy porządek

Na początku lat 70 ubiegłego wieku do tworzenia prostych planów wykorzystywano również pojęcie liniowego porządku (*total order*) sekwencji akcji. W tym podejściu, ochrzonym mianem **planowanie liniowe** (*linear planning*) dla każdego z celów próbowano utworzyć odpowiedni podplan, który go osiągał. Następnie ów plan łączono przy pomocy odpowiedniego porządku akcji w jeden spójny plan. Ze względu na wady tego podejścia, jak długi czas formowania planu oraz możliwość wystąpienia konfliktów między odpowiednimi celami planu, co na początku nie było takie oczywiste, zamiast liniowego porządku zaczęto wykorzystywać porządek częściowy (*partial order*), którego przykładem jest omawiany **GRAPHPLAN**.

1.2.4 ADL i PDDL

Action Description Language (w skrócie **ADL**) zostało zaproponowane przez pana Edwina Pednault'a w 1987 roku jako usprawnienie języku opisu STRIPS. Ów system automatycznego planowania został zaprojektowany z myślą o przyszłej implementajci w nowoczesnych na tamten czas robotach. Główną ideą stojącą za utworzeniem tego języka opisu problemów było zauważenie, iż język STRIPS nie potrafi poradzić sobie z sytuacjami, gdy powstanie danego efektu jest niedeterministyczne. Dodatkowym aspektem, którego wprowadzenie znacznie poprawiło wydajność metodologii ADL względem STRIPS było wykorzystanie zasady **otwartego świata**, która mówi, iż rzeczy nieokreślone w świecie są uznawane jako *nieznane*, a nie jako fałszywe, jak to było w STRIPS (**zasada zamkniętego świata**). Dodatkowo, ostateczne cele w języku STRIPS mogły być jedynie przedstawione jako koniunkcje stanów (np. Dobra i Tania, np. w kontekście restauracji), czyli jako cele bezwarunkowe. W nomenklaturze ADL cele można przedstawiać również przy pomocy alternatyw, to znaczy nie wszystkie stany muszą być spełnione, mogły istnieć grupy stanów, w których spełnienie tylko jednego było wystarczające (np. Dobra i (Tania lub Blisko)).

Planning Domain Definition Language (w skrócie **PDDL**) było próbą ustandaryzowania języków wykorzystywanych do planowania przy pomocy komputerów silnie inspirowaną przez swoich poprzedników-STRIPS i ADL. Utworzeniem takiego standardu zajął się Pan Drew McDermott wraz ze współpracownikami z **Yale University**. Podstawową ideą stojąca za PDDL jest całkowite odseparowanie metod rozwiązywania problemów od środowisk, w których się znajdują. Techniki rozwiązywania problemów związanych z logistyką (ustalanie najkrótszej drogi) powinny być również aplikowalne do rozwiązywanie problemów związanych z produkcją (minimalizacja zużycia materiału). Pozostałe aspekty są porównywalne do języka STRIPS- istnienie warunków początkowych, zbioru operatorów oraz zbioru złożonego z celow do osiągnięcia.

1.2.5 Nowoczesne rozwiązania

TO-DO

1.2.6 Miejsce GRAPHPLAN'u

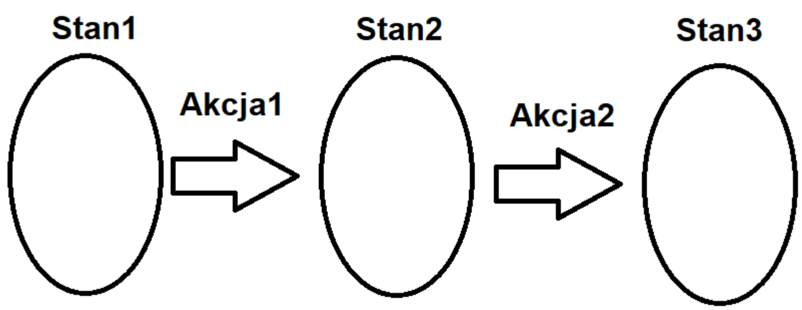
Między powstaniem metodologii STRIPS a jej rozszerzeniami w postaci ADL lub PDDL powstawały również inne podejścia, w tym silnie bazujący na grafach, własnościach częściowego porządku i ich możliwościach algorytm o wdzięcznej nazwie **GRAPHPLAN**.

Rozdział 2

GRAPHPLAN

2.1 Wprowadzenie

GRAPHPLAN jest algorytmem do planowania akcji działający w dziedzinie zdefiniowanej przez język STRIPS, dodatkowo bazuje na paradygmacie, który autorzy algorytmu określają jako "graf planujący"[2].



Rysunek 2.1: Najogólniejsza forma grafu planującego. Składa się on z węzłów, zwanych stanami oraz krawędzi zwanych akcjami. Docelowo poszczególne stany oraz akcje są parami różne, jednak mogą zajść sytuacje, gdy powtórzenie któregoś z komponentów będzie wymagane do uzyskania odpowiedniego celu.

Pierwotną ideę grafu planującego przedstawiono na obrazku 2.1. W trakcie dalszego omawiania metodologii GRAPHPLAN powyższa rycina będzie pojawiała się ponownie z coraz to większym poziomiem szczegółowości. Ze względu na fakt, iż Graphplan opiera się na języku STRIPS musi mieć jasno zdefiniowane: stan początkowy, akcje oraz cel, który pragniemy uzyskać. Dzięki swojej strukturze Graphplan w swojej naturze podobny jest do programowania dynamicznego.

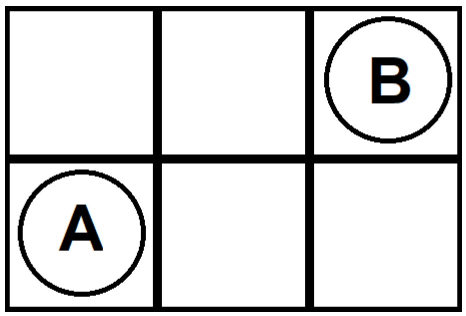
2.2 Warunki początkowe

Definicja 2.1 ***Zasada zamkniętego świata*** *- Zasada, wedle której pojęcia, które nie są ściśle opisane w świecie są nieprawdziwe.*

GRAPHPLAN, w odróżnieniu od człowieka, musi być w posiadaniu całej wiedzy o świecie, aby móc rozpocząć działanie. Przez całą wiedzę o świecie rozumie się posiadanie informacji na temat każdego obiektu oraz jego stanu. Wiąże się to ściśle z faktem, iż GRAPHPLAN operuje zgodnie z definicją 2.1. Przed dalszą częścią pracy należy dokonać pewnego wyróżnienia. Słowo **stan** pojawia się w dwóch znaczeniach: stan jako pojedyncza informacja o obiekcie w świecie (Przykład: klocek B na stole numer 3) oraz stan, jako zbiór wszystkich takich informacji w danym momencie czasu. Z tego względu wprowadzono nowe pojęcie- "Poziom stanów", które należy stosować jako oznaczenie wszystkich informacji o świecie w danym momencie.

Definicja 2.2 ***Poziom stanów*** *- Zbiór informacji o stanach wszystkich obiektów w świecie w danej jednostce czasu t*

Szczególnym poziomem stanów jest poziom oznaczany jako pierwszy i nazywany **Warunkami początkowymi**, którego poprawne zdefiniowanie jest kluczowym aspektem w kontekscie uzyskania poprawnego wyniku przez algorytm. Analizując ponownie przykład 1.1 mylnym jest myśleć, iż jedyną informacją, jaką algorytm powinien posiadać o świecie jest pobyt klocka A na lewej platformie. Również istotną informacją jest brak klocka na platformie prawej, czyli informacja, ze jest on *pusty*. Mimo poczucia nadmiarowości tej informacji, w dalszej części pracy wyjaśni się, dlaczego ta informacja jest niezbędna do uzyskania poprawnego wyniku. Przykład świata przedstawionego oraz skonstruowanego dla niego stanu początkowego:



Rysunek 2.2: Przykładowy moment startowy przyszłego planu. Za pomocą okręgów oznaczono roboty, natomiast poprzez kwadraty oznaczone są kafelki- miejsca, po których mogą poruszać się roboty.

Na powyższym przkładzie, zgodnie z ideą Graphplanu wyszczególniamy 6 stanów początkowych. Dodatkowo należy doprecyzować pojęcie bycia robota na danym kafelku. Wykonano to przy pomocy dwuargumentowej relacji *na*, która jako pierwszy argument przyjmuje sygnaturę robota, a na drugim- numer kafelka. Na potrzeby przykładu ustalono, iż numerowanie odbywa się rzędami od lewej do prawej. Zgodnie z tymi ustaleniami pozycję robotów A i B możemy określić w następujący sposób: $na(A,4)$ oraz $na(B,3)$. Również pustość kafelków należy sformalizować wprowadzając relację jednoargumentową o nazwie *pusty*, która przyjmuje jako argument numer pustego kafelka. Reasumując, zbiorem stanów początkowy dla analizowanego przykładu 2.2 jest:

$$\{pusty(1),pusty(2),na(B,3),na(A,4),pusty(5),pusty(6)\} \tag{2.1}$$

2.3 Akcje

Posiadając dobrze określony stan początkowy następnym krokiem będzie zdefiniowanie akcji. Zgodnie z 1.2 oraz wzmiance o akcjach w planerze STRIPS, akcja musi składać się z trzech komponentów:

- Czynności
- Warunków zajścia
- Efektów zajścia

Z tego powodu każdą z akcji będziemy traktować jako trójkę

$$A = (C,W,E) \tag{2.2}$$

gdzie każda z liter odpowiada pierwszej literze wyżej wymienionego pojęcia. W skład efektów wchodzą dwa pojęcia wprost z terminologii STRIPS- dodające i usuwające. Dzięki takiemu podziałowi łatwiejszym będzie zachowanie silnego podziału między przyczynami a efektami akcji. Jedyną czynnością, którą należy brać pod uwagę w ramach 2.2 jest czynność *ruch*, którą definiujemy jako trzyargumentową relację:

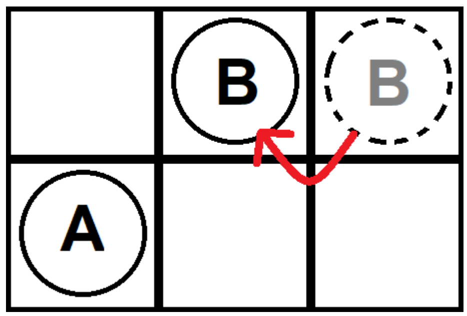
$$ruch(R,S,D) \tag{2.3}$$

, gdzie R odpowiada robotowi, który musi się przemieścić z kafelka oznaczonego literą S (kafelek startowy) na kafelek oznaczony literą D (kafelek docelowy).

Następnymi składowymi są odpowiednio *Warunki* jak i *Efekty*. Warunki traktujemy jako zbiór wszystkich stanów, które muszą być prawdziwe w danej jednostce czasu. Jeśli choć jeden stan nie jest spełnialny, opisywana akcja nie może zostać wykonana w danym ruchu. Efekty natomiast definiujemy jako następującą parę:

$$E = (D,U) \tag{2.4}$$

gdzie D oznacza efekty dodające, a U- efekty usuwające.



Rysunek 2.3: Obrazowe przedstawienie ruchu robota B z kafelka 3 na kafelek 2

Niech rozpatrywaną akcją będzie przemieszczenie robota B z pozycji 3 na pozycję 2, przedstawiona na 2.3. Biorąc pod uwagę, iż stanem początkowym jest 2.1 Warunkami zajścia zdarzenia będą: $na(R,S)$ oraz $pusty(D)$. Dla efektów natomiast sytuacja wygląda następująco: efektami dodającymi są $na(R,D)$ oraz $pusty(S)$, które informują o tym, iż klocek wykonał ruch z kafelka S na kafelek D, a efektami usuwającymi $\sim na(R,S)$ oraz $\sim pusty(D)$, które informują o tym, iż kafelek S został zwolniony, oraz robot nie znajduje się już na kafelku S. Przy pomocy matematycznego symbolu negacji wyrażono nieprawdziwość danego stanu. Ze względów estetycznych oraz ułatwiających analizowanie pracy negacje stanów w dalszej części pracy wyrażono również przy pomocy polskiej partykuły przeczącej **nie**. Dla przykładu pojęcia $\sim pusty(1)$ oraz $niepusty(1)$ z perpsektywy wprowadzonej terminologii są tożsame.

Posiadając następującą wiedzę poniżej zdefiniowano jedyną akcję znajdująca się w prezentowanym przykładzie:

$$A = (ruch(R,S,D),\{na(R,S),pusty(D)\},\{na(R,D),pusty(S), \sim na(R,S), \sim pusty(D)\}) \tag{2.5}$$

Podstawiając za $R = B$, $S = 3$, a $D = 2$ otrzymujemy następującą akcję:

$$A = (ruch(B,3,2),\{na(B,3),pusty(2)\},\{na(B,2),pusty(3), \sim na(B,3), \sim pusty(2)\}) \tag{2.6}$$

Analogicznie można zdefiniować ruch na kafelek numer 6, oraz dwa ruchy dla robota o sygnaturze A.

2.3.1 Typy akcji

Definicja 2.1 znajduje również swoje odzwierciedlenie w akcjach. Niech rozpatrywanym przykładem będzie wciąż przykład 2.2. W pierwszym kroku wykonano akcję $ruch(B,3,2)$.Z persepktywy człowieka jest to wystarczająca informacja, aby móc wydedukować, co we wskazanym etapie generowania planu dzieje się z robotem **A**. Otóż robot A w pierwszym ruchu zostaje na tym samym kafelku. Jednakże ze względu na zamkniętość świata algorytmu należy go również poinformować go o tym, w jakim stanie po pierwszym kroku ma znajdować się robot A. Wykonywane jest to przy pomocy akcji, zwanych **akcjami potrzymującymi**.

Definicja 2.3 ***Akcja podtrzymująca*** - *Akcja, która przenosi stan obiektu w czasie t nienaruszonym do poziomu stanów w czasie $t + 1$*

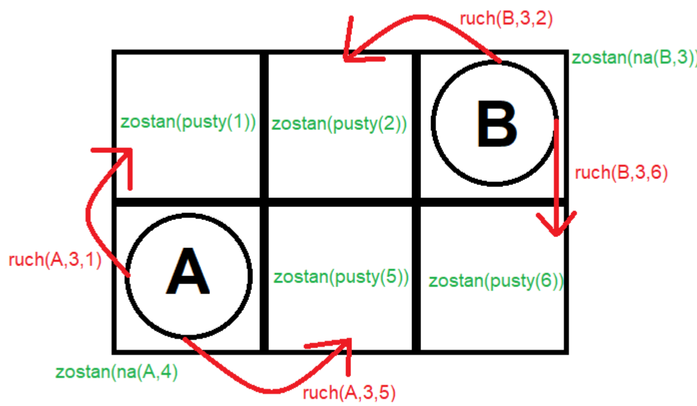
Akcjami podtrzymującymi należy również informować świat o stanach kafelków, które nie brały udziału w akcji robota B. Przykładem takiego jest kafelek 1, który w stanie początkowym, jak i w stanie następnym ciągle zachowuje swój stan jako pusty. Akcje podtrzymujące oznaczono słowem kluczowym **zostań** Ponadto akcja typu *ruch*, które aktywnie zmienia stan świata w dalszej części pracy otrzyma miano **akcji aktywnej**.

Definicja 2.4 ***Akcja aktywa*** - *Akcja, która zmienia stan obiektu między stanami w czasie t i $t+1$.*

Posiadając powyższy podział akcji poniżej przedstawiono pełen zbiór akcji w pierwszym kroku algorytmu. Wartym odnotowania jest, iż ze względów estetycznych podawawnie akcji podtrzymujących będzie często pomijane, jednakże nie można zapomnieć o ich występowaniu oraz o ich kluczowej roli w generowaniu precyzyjnego planu.

$$Akcje = \{zostan(pusty(1)), zostan(pusty(2)), zostan(pusty(5)), zostan(pusty(6)), zostan(na(B,3)), zostan(na(A,4)), ruch(B,3,2), ruch(B,3,6), ruch(A,4,1), ruch(A,4,5)\}$$

Należy również nadmienić, iż podobnie jak w stanach, dla akcji wprowadza się pojęcie **Stanu akcji**, które funkcjonuje jako zbiór składający się ze wszystkich możliwych akcji do wykonania w danej jednostce czasu.



Rysunek 2.4: Obrazowe przedstawienie wszystkich akcji w pierwszym kroku algorytmu, akcje opisane przy użyciu czcionki o kolorze zielonym symbolizują akcje podtrzymujące, natomiast o kolorze czerwonym- akcje aktywne

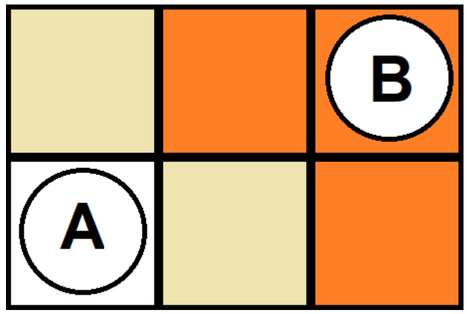
2.4 Definiowanie świata

Zdecydowanie najtrudniejszym aspektem modelowania świata jest dokładne przedstawienie wszystkich zależności, jakie algorytm musi znać, aby mógł bezbłędnie wnioskować w prezentowanej przestrzeni. Analizując przykład 2.2 dokonano przedstawienia schematu generowania poziomu stanów początkowych oraz poziomu akcji. Zadając pytanie algorytmowi *Jakie działania należy podjąć, aby jak najszybciej przesunąć robota A z klocka 4 na klocek 6* algorytm odpowie w następujący sposób: $ruch(A,4,6)$, jednakże przyglądając się uważnie przedstawionemu światu zauważono, iż taki plan byłby realny jedynie, gdyby akcja *ruch* oznaczała latanie- wtedy faktem jest, iż istnieje możliwość pominięcia kafelka 5 podczas przemieszczenia. Niespójnośc ta pojawiła się z faktu, iż w żadnym momencie nie określono w jaki sposób dokładnie funkcjonuje czynnośc oznaczona jako *ruch*. Wedle definicji 2.5 wszystkie warunki, aby robot mógł przemieścić się z kafelka 4 na 6 zostały spełnione.

Naprawić ten problem można przy pomocy uściślenia wykonywanej czynności. Do tego potrzebne będzie wprowadzenie relacji *sąsiad*, która przyjmuje dwa argumenty- numery kafelków, które ze sobą sąsiadują. Zakładając, iż sąsiadujące kafelki to takie, które mają wspólną ścianę, sąsiadami kafelka 1 są kafelki: 2 oraz 4. Dla pozostałych przestrzeni dokonano analogicznego wygenerowania listy sąsiadów. Dzięki tej operacji, akcję ruch zdefiniowano w następujący sposób:

$$ruch(R,S,D)\text{:-}sasiad(S,D) \tag{2.7}$$

Czynność ruchu w powyższym przypadku została określona zgodnie z semantyką języka programowania **PROLOG**. Należy to rozumieć w następujący sposób: po lewej stronie znaku **:-** znajduje się *konkluzja*, natomiast po prawej- *przesłanka*. Naturalnym odczytem przedstawionej sytuacji będzie zdanie *"Jeśli kafelki S i D są sąsiadami to możliwym do wykonania jest ruch między tymi kafelkami"*



Rysunek 2.5: Graficzne przedstawienie relacji sąsiedztwa dla kafelka numer 4. Kolorem kremowym oznaczono miejsca, z którymi sąsiaduje, natomiast pomarańczowym- te, z którymi nie sąsiaduje. Oznacza to, iż do kafelków pomarańczowych nie można dostać się wykonując jeden ruch

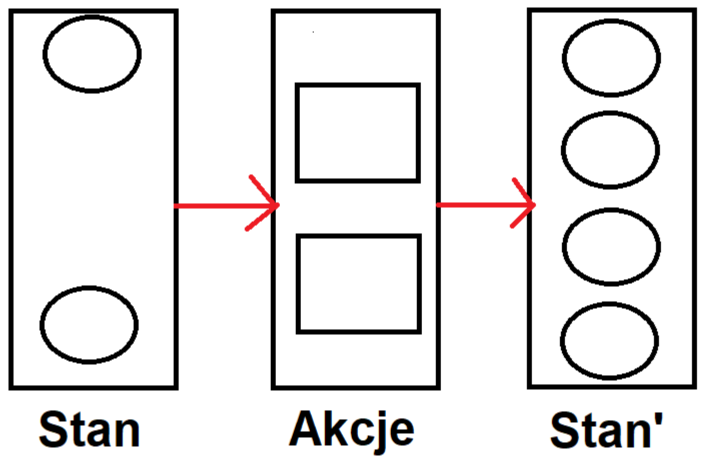
Taka definicja ruchu pozwala w skuteczny sposób oddać intuicję, która wynika z poglądowego obrazka przedstawiającego rozpatrywany świat. Niewystarczającym jest określenie relacji $sasiad(1{,}2)$, gdyż wedle tego faktu kafelek pierwszy sąsiaduje z kafelkiem drugim, jednakże nie oznacza to, iż kafelek drugi sąsiaduje z kafelkiem pierwszym. Z tego dla każdej pary kafelków A i B do zbioru sąsiadów należy dodać dwie relacje: $sasiad(A{,}B)$ oraz $sasiad(B{,}A)$. Inna sytuacja byłaby, gdyby ruch był możliwy jedynie w jedną stronę, jednakże tutaj taki stan rzeczy nie występuje.

Oprócz definicji sąsiedztwa należy również zabezpieczyć się przed sytuacją, gdy dwa roboty będą chciały w tym samym momencie znaleźć się na tym samym kafelku. Dodatkowo kafelek A nie może znajdować się w dwóch stanach jednocześnie: $pusty(A)$ oraz $na(R{,}A)$, gdzie R oznacza dowolnego robota. Wymodelowanie tych ograniczeń jest prostym, lecz istotnym zadaniem implementując przedstawianą metodologię, które zostanie poruszone w ramach przedstawiania pojęcia "wzajemnego wykluczania".

Odpowiednie wymodelowanie świata wedle wzorców języka STRIPS może wydawać się trudnym oraz żmudnym zajęciem, jednakże przebrnąwszy przez ten etap algorytm jest zwarty i gotowy do generowania planów dla wprowadzonych celów.

2.5 Warstwy grafu

Składowe planu grafującego można podzielić na dwa typy: jeden, wyszczególniony jako poziomy stanów oraz drugi- poziomy akcji. Aby lepiej uwidocznić zależność poziomu akcji od warunków początkowych oraz zależność kolejnego poziomu stanów od efektów akcji graf planujący z 2.1 ulegnie lekkiej modyfikacji.

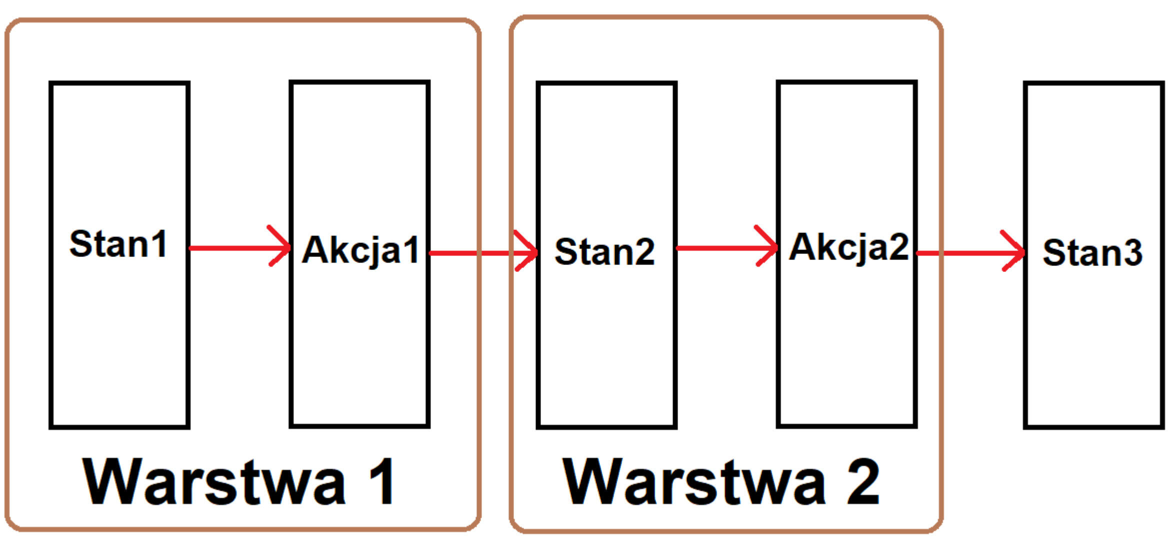


Rysunek 2.6: Modyfikacja do grafu planującego wprowadzona w poprzednich podrozdziałach. Wprowadzono zależności akcji od stanu poprzedniego, oraz stanu następnego od akcji.

Następnym krokiem będzie wprowadzenie definicji **Warstwy grafu**

Definicja 2.5 ***Warstwą*** *- grafu nazywamy połączenie poziomu stanów oraz wynikającego z niego poziomu akcji*

Przez i-tą warstwę grafu oznaczono stan świata w i-tym momencie czasu. Ze względu na charakterystykę planera czas traktowany jest w sposób dyskretny- każda ze zdefiniowanych akcji zajmuje zawsze tyle samo czasu oraz zawsze kończy się tym samym efektem. Poziomy stanów jak i poziomy akcji należące do tej samej i-tej warstwy nazywamy poprzez dołączenie do ich nazwy numera, odpowiadającego obecnej iteracji algorytmu iteracji algorytmu.



Rysunek 2.7: Przedstawienie sposobu wyznaczania warstw w algorytmie planującym. Poziom stanów oraz akcji wchodzący w skład danej warstwy wyróżniony jest poprzez konkatenację nazwy oraz liczby symbolizującej numer warstwy.

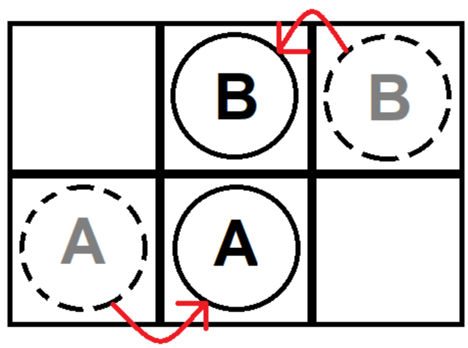
Dzięki wprowadzeniu definicji warstwy po wygenerowaniu planu natychmiast wiadomo ile kroków należy wykonać, a co za tym idzie- ile czasu należy poświęcić, aby osiągnąć ustalony cel.

2.6 Równoległość

W poprzednich paragrafach na pojedynczym poziomie akcji rozpatrywano conajwyżej jedną akcję aktywną, jednakże główną siła GRAPHPLANU, jest możliwość jego reprezentacji jako częściowego porządku. To pozwala w prosty sposób wprowadzić pojęcie równoległości między akcjami.

Definicja 2.6 *Dwie (lub więcej) akcje mogą zostać wykonane* ***równolegle****, gdy po ich wykonaniu w tej samej warstwie i, warstwa $i+1$ nie będzie zawierała w sobie żadnych sprzeczności.*

Przyglądając się rysunkowi 2.2 od razu należy zauważyć, iż w stanie początkowym ruch robota A nie wpływa w żaden sposób na otoczenie robota B. Z tego względu dwie przykładowe akcje $ruch(A,4,5)$ oraz $ruch(B,3,2)$ wręcz należy wykonać w tej samej jednostce czasu, czyli w pierwszym poziomie akcji.



Rysunek 2.8: Przykład możliwości zastosowania równoległości w planowaniu działania. Należy zauważyć, iż ruchy robotów w żaden sposób ze sobą nie kolidują.

Po wprowadzeniu definicji równoległości, również definicja kroku musi ulec ewolucji.

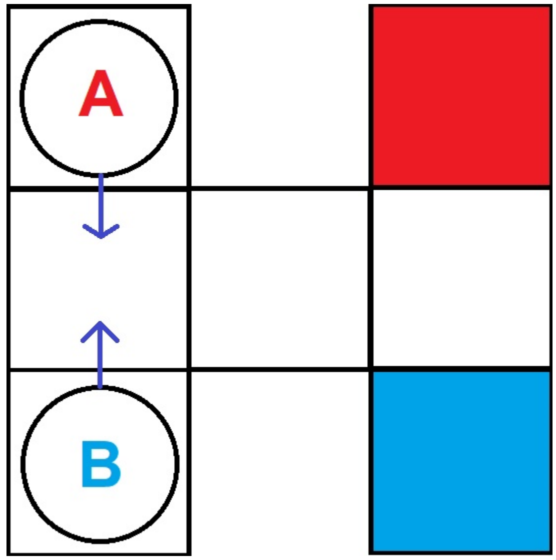
Definicja 2.7 ***Krokiem*** *algorytmu nazywamy zbiór wszystkich możliwych do realizacji akcji w danej warstwie.*

Dzięki naniesionej poprawce pozbywamy się łatki mówiącej o tym, iż pojedyncza zmiana stanu jest ściśle związana z jedną akcją.W tym miejscu należy zauważyć, iż poprzednie analizy zawierały spore uproszczenie, gdyż akcje podtrzymujące również należą do kroku algorytmu, co zostało zaprezentowane w trakcie kreowania pierwszej warstwy akcji dla rozpatrywanego przykładu.

Istnieją przypadki, gdy podmiot realizujący plan nie jest w stanie wykorzystać benefitów płynących z możliwości dokonywania akcji równolegle, na przykład ze względu na ograniczone zasoby. GRAPHPLAN również rozwiązuje ten problem, pozwalając przekształcać swój częsciowy porządek na porządek liniowy w dość dowolny sposób. Otóż akcje z danej warstwy mogą być wykonywane w dowolnej kolejności, gdyż w żaden sposób ze sobą nie kolidują. Z persepktywy algorytmu najważniejszym jest, aby stan świata i i $i+1$ był zgodny z informacjami jakie posiada o świecie. Zgodnie z przykładem 2.9 widać, iż to, czy akcja $ruch(A,4,5)$ zostanie wykonana przed akcją $ruch(B,3,2)$ lub to, czy akcja $ruch(B,3,2)$ odbędzie się prxed $ruch(A,4,5)$- efekt końcowy jest identyczny.

2.6.1 Wzajemne wykluczanie

Twórcy algorytmy wprowadzając równoległość, zdawali sobie sprawę z mocy tego podejścia. Dzięki temu ogromna liczba planów ulega redukcji jeśli chodzi o wymagany czas wykonania. Jednakże równoległość wprowadza kolejny istotny problem w prezentowanym świecie, a mianowicie- co, gdy wykonanie dwóch akcji będzie wprowadzało sprzeczność w następnym poziomie świata? Sytuacja ta została przedstawiona na poniższym rysunku



Rysunek 2.9: Przykład świata, w którym wprowadzenie równoległości dla pierwszej warstwy algorytmu jest niemożliwe. Dwa roboty próbują przejść na ten sam kafelek w tej samej jednostce czasu, co z perspektywy kafelka powoduje konflikt. Odpowiednio kolorami: czerwnoym i niebieskim oznaczono roboty, oraz kafelki, które są ich celem.

Zmusiło to twórców algorytmu do wporwadzenia pojęcia **relacji wykluczania**. O akcjach wzajemnie się wykluczających (ang. actions mutually exclusive, **mutex**) wspomiano w ramach definiowana świata. Pojęcie ów można określić w następujący sposób:

Definicja 2.8 ***Relacją wzajemnie wykluczającą się*** *- jest relacją między akcjami(stanami), która informuje o tym, iż nie istnieje plan taki, aby dwie wybrane akcje(stany) mogły być prawdziwe w tej samej jednostce czasu t*

Przykładem stanów wykluczających jest para: $pusty(1)$ oraz $\sim pusty(1)$. Kafelek nie może być pusty jak i niepusty jednocześnie. Przykład dwóch akcji wykluczających przedstawiono na rysunku 2.9. Należy zauważyć, iż wzajemne wykluczanie się jest rozpatrywane warstwowo. Kolejnym naturalnym krokiem jest ustalenie, kiedy akcje oraz stany są ze sobą w relacji wykluczającej.

Wykluczanie się stanów

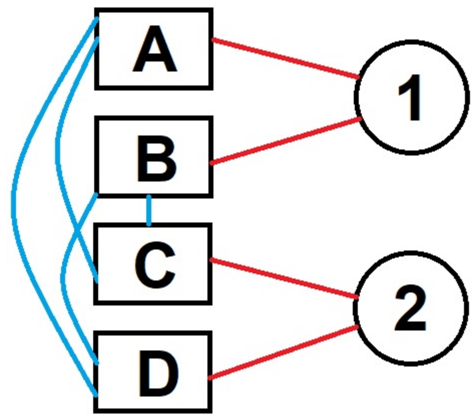
Dwa stany są ze sobą w relacji wzajemnie wykluczającej w dwóch następujacych przypadkach:

1. Negacja- przypadek, w którym jeden ze stanów jest negacją drugiego

2. Niespójne powstanie- wszystkie akcje z poprzedniej warstwy, które prowadza do utworzenia ów stanów są ze sobą parami w relacji wykluczającej

Ze względu na naturalność pojęcia negacji zbędnym jest wprowadzenie większego przykładu niż przedstawienie, iż dla każdego obiektu istnieje para stanów znajdujących się w relacji wykluczającej. Niech za przykład posłuży pustość kafelka. Kafelek nie może być jednocześnie pusty, jak i niepusty, co sprowadza się, iż stan $pusty(kafelek)$ jak i $\sim pusty(kafelek)$ są niemożliwe do zawarcia w jednej warstwie planu.

Dla kontrastu przedstawienie przypadku, w którym zachodzi **niespójne powstawanie** jest trudniejsze i wymaga użycia przykładowej ilustracji:



Rysunek 2.10: Urywek planu przedstawiający sytuację, w której dwa stany powstają w sposób niespójny. Zgodnie z założeniami akcje oznaczone są przy pomocy prostokątów, stany- okręgów, miedzy akcjami a stanami czerwone linie symbolizują które stany są efektami których akcji, natomiast linie niebieskie między akcjami symbolizują powstałe między nimi *mutexy*

Zgodnie z nienaturalnym przykładem 2.10 należy zauważyć, iż stany 1 oraz 2 nie mogą znajdować się jednocześnie w planie ze względu na to, iż każda z akcji, która generuje stan 1 (B,C) znajduje się w relacji wzajemnie wykluczającej z każdą akcją, która generuje stan 2 (C,D). Oznacza to, iż ów dwa stany powstają w sposób **niespójny**.

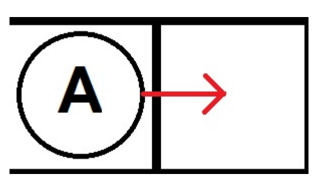
Z reguły ten typ wykluczeń występuje po większej liczbie kroków dla bardziej rozbudowanych światów jak i planów.

Wykluczanie się akcji

Dwie akcje mogą być ze sobą w relacji wzajemnie wykluczającej w trzech następujących przypadkach:

1. Niespójny efekt- przypadek, w którym zbiór efektów jednej z akcji jest negowany przez zbiór efektu drugiej
2. Przeszkadzanie - przypadek, w którym jedna z akcji usuwa warunki zajścia akcji drugiej
3. Konkurencyjne potrzeby- przypadek, w którym warunki zajścia akcji są ze sobą w relacji wykluczającej.

Poniższe przykłady w obrazowy sposób przedstawiają każdy w wyżej wymienionych przypadków:



Rysunek 2.11: Przykład wykluczania się akcji aktywnych z podtrzymującymi

Na pierwszy rzut oka wydawać by się mogło, iż niemożliwym jest wygenerowanie dwóch akcji znajdujących się w relacji wykluczania dla przykładu 2.11, jednakże istnieją dwie takie pary:

$$zostan(na(A,lewy)),ruch(A,lewy,prawy) \tag{2.8}$$

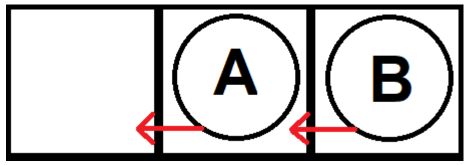
oraz

$$zostan(pusty(prawy))orazruch(A,lewy,prawy) \tag{2.9}$$

Wynika to z faktu, iż zbiór efektów jednej akcji z powyższych par jest negowany przez drugi. Zbiorem efektów akcji *zostan*(*na*(*A*,*lewy*)) jest *na*(*A*,*lewy*), jednakże zbiorem efektów *ruch*(*A*,*lewy*,*prawy*) jest ciut większy zbiór:

$$na(A{,}prawy), \sim na(A{,}lewy), pusty(lewy), \sim pusty(prawy) \tag{2.10}$$

Widać, iż efekt $\sim na(A{,}lewy)$ jest negacją efektu $na(A{,}lewy)$. Akcje podtrzymujące i aktywne względem tego samego obiektu zawsze sobą ze sobą w relacji wykluczającej ze względu na **niespójny efekt**.



Rysunek 2.12: Przykład świata, w którym każdy z robotów próbuje przesunąć się na sąsiadujący z lewej strony klocek. Uwidoczniona sytuacja jest przykładem wykluczających się akcji

Idea przedstawiona na rysunku 2.12 jest ciekawym przykładem, pozwalającym zrozumieć dokładniej definicję słowa **równoległość** odnośnie planów generowanych przez GRAPHPLAN. Gdyby roboty poruszając się z tą samą szybkością ruszyły w tym samym momencie to wykonanie ów sekwencji akcji byłoby w zupełności możliwe, jednakże w sekcji 2.6 wspomniano, iż każdy z wygenerowanych planów musi być możliwy do przedstawienia w postaci porządku liniowego, czyli plan równoległy może zostać przerobiony w dowolny sposób na plan, w którym każda z akcji wykonywana jest po kolei. Zgodnie z omawianym przykładem sprowadzenie ów planu do dowolnej postaci liniowej jest niemożliwe, gdyż istnieje konfiguracja, w której to najpierw robot B miałby wykonać ruch, jednakże jest to niemożliwe ze względu na obecność robota A na kafelku będącym jego celem podróży. Dodatkowe wprowadzenie ograniczeń na konwersję planów równoległych na liniowe jest bardziej skompilowanym zagadnieniem, którego omówienie nastąpi w sekcji odpowiedzialnej za możliwe rozwinięcia algorytmu.

Z powyższego opisu wynika, iż omawiane relacje ruchu są ze sobą w relacji wzajmenie wykluczającej z powodu **przeszkadzania**. Klocek A do wykonanaia ruchu potrzebuje znajdować się na kafelku środkowym, co oznacza, iż kafelek środowy musi byc niepusty, jednakże klocek B potrzebuje, aby ów klocek był pusty. Relacja wykluczająca między tymi stanami bezpośrednio generuje wykluczanie się tych akcji.

Ponadto, omawiane akcje w ilustracji 2.12 znajdują się w relacji wykluczającej z powodu **Konkurencyjnych potrzeb**. Robot B przy przesunięciu **na** środkowy kafelek wymaga, aby on był pusty, natomiast robot A przy przesunięciu **z** środkowego klocka musi się na nim znajdować, czyli kafelek musi być niepusty. Stany pusty i niepusty względem kafelka są w oczywistej relacji wykluczającej co automatycznie generuje wykluczenie się wspomnianej pary akcji. Przykład ów przedstawia, iż jedna para akcji może być ze sobą w relacji wykluczającej z kilku powodów, jednakże algorytm rozpatrując parę akcji podchodzi do tego w sposób binarny, co oznacza, iż patrzy jedynie czy znajdują się w relacji wykluczającej czy nie, nie interesuje go liczba sposobów, na które ów relację można utworzyć.

Należy zauważyć, iż dzięki wprowadzeniu relacji wzajemnego wykluczania pozbyto się porównań wielu akcji, których zajście jest niemożliwe w tej samej warstwie grafu. Koszt jaki został poniesiony ze względu na zapamiętywanie dodatkowych informacji o relacjach między akcjami jest często zaniedbywalny, ze względu na ogrom benefitów w postaci mniejszej liczbie sprawdzeń, a co za tym idzie- szybsze działanie algorytmu.

2.7 Wyszukiwanie planu

Zdefiniowawszy wszystkie niezbędne elementy planera należy wskazać, w jaki sposób przy posiadaniu całej wiedzy wygenerowanej przez graf planujący. Wykonywane jest to w następujący sposób:

1. Rozpoczynając od stanu początkowego, zgodnie z opisanymi w poprzednich sekcjach metodami, odbywa się generowanie kolejnych warstw grafu, biorąc pod uwage informację o **mutexach** między akcjami oraz stanami

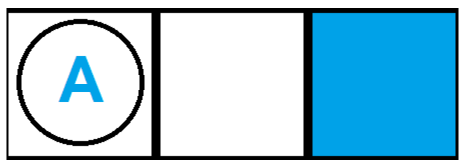
2. Dla każdej nowo utworzonej warstwy i dokonywane jest sprawdzenie, czy wszystkie założenia z celu nie znajdują się w ów poziomie stanów. Jeśli odpowiedź jest negatywna, odbywa się dalsze generowanie planu zgodnie z 1. Jednak, gdy wszystkie stany celu znajdują się na poziomie i dochodzi do generowania planu.

3. Dla każdego stanu z celów na poziomie i dochodzi do wybrania akcji, dzięki której został on wygenerowany. Ów operacja dokonywana jest dla każdego stanu. Jeżeli dobrane akcje są ze sobą w relacji wykluczającej, należy spróbować innej kombinacji akcji. Jeśli wszystkie dobory akcji zawiodą należy wygenerować kolejną warstwę grafu i ponownie, w warstwie $i + 1$, rozpocząc cały proces

4. Jeśli jednak istnieje dobór akcji taki, że nie występuje między nimi relacja wykluczania, należy dla każdego stanu znajdującego się w zbiorze warunków zajścia wspomnianych akcji wykonać procedurę z kroku 3.

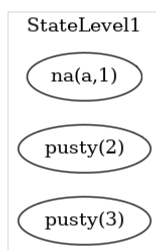
5. Jeśli dojdzie do niepowodzenia na którymkolwiek z etapów powrotu do stanu wyjściowego algorytm podejmuje próbuję odpowiedniego dobrania akcji na ostatnio sprawdzonym poziomie. Brak niepowodzenia na ścieżke powrotu od warstwy i do warunków początkowych świadczy o tym, iż istnieje sekwencja akcji pozwalająca otrzymać stany zdefiniowane z celu w określonym przez stany początkowe świecie, co oznacza, iż jest możliwym utworzenie odpowiedniego **planu**.

Aby precyzyjnie przedstawić działanie algorytmu w następnej części programu przeprowadzono rozbudową analizę konkretnego przykładu:



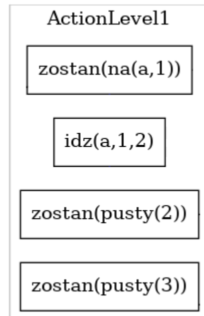
Rysunek 2.13: Sytuacja początkowa świata, dla którego odbędzie się przykładowe generowanie planu. Kolorem niebieskim zaznaczono kafelek docelowy robota

2.13 przedstawia świat, w którym robot A z kafelka 1 próbuje przedostać się do kafelka 3 (Kafelki ponownie numerowane są od lewej strony do prawej).



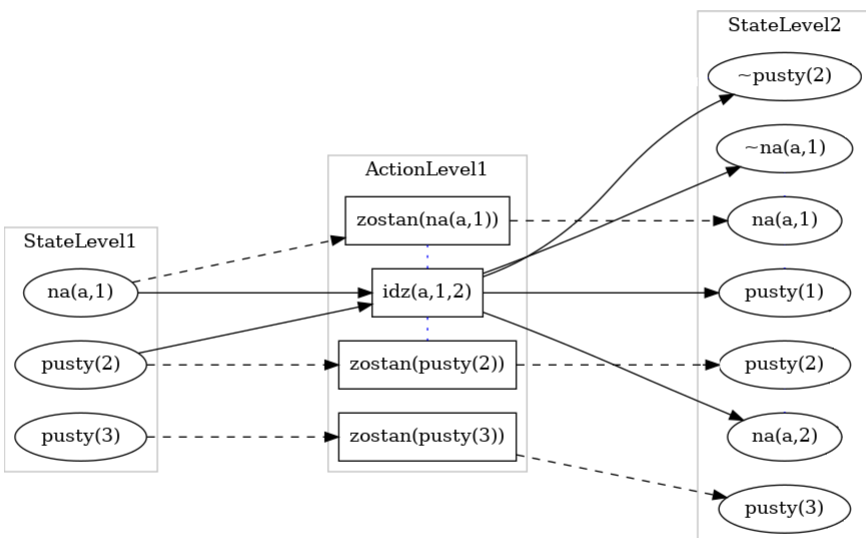
Rysunek 2.14: Warunki początkowe omawianego świata. Obrazki ułatwiające analizę przykładu w całości zostały wykonane przez oprogramowanie utworzone na rzecz pracy.

Ze względu na fakt, iż w języku programowania **PROLOG**, w którym implementowany jest algorytm, zmienne są oznaczane przy pomocy wielkiej litery, a stałe przy pomocy małej, wielkość litery w oznaczaniu nazwy robota nie ma istotnego znaczenia. Zwyczajowo w opisie robot będzie określany przy pomocy wielkiej litery (np. **A**) natomiast w wygenerowanych przez program grafach przy pomocy małej **a**. W powyższy sposób zaprezentowano stan początkowy świata. Wszystkie definicje relacji takich jak **na**, **idz**, **zostan**, czy **pusty** wprowadzono w podrozdziałach 2.2 i 2.3. W wygenerowanych przez oprogramowanie grafach **poziomy stanów** określane są poprzez swój angielski odpowiednich **StateLevel**. Podobnie z **poziom akcji** i **ActionLevel**.



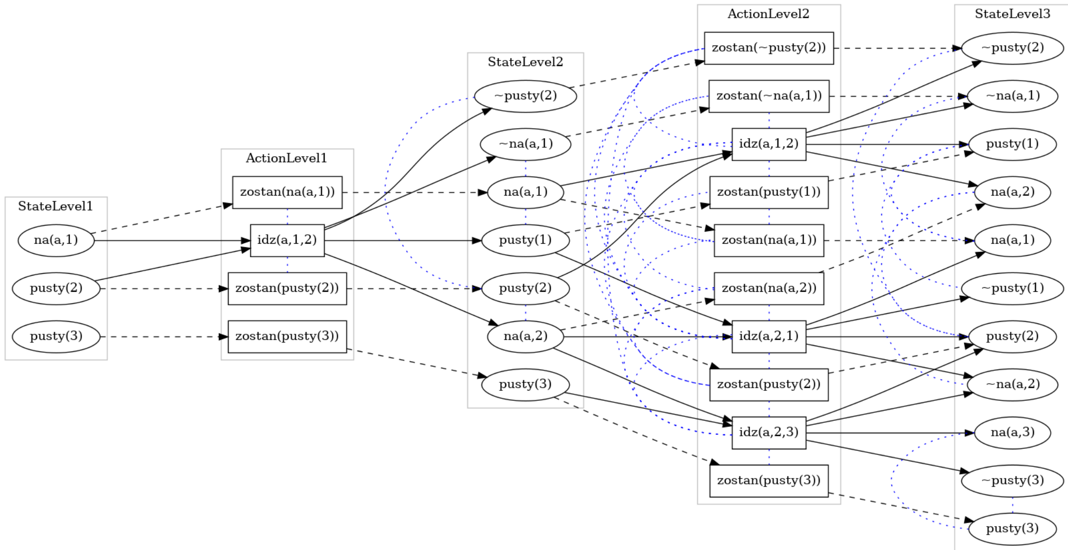
Rysunek 2.15: Wygenerowane akcje na podstawie warunków początkowych

Następnie przystąpiono do wygenerowania wszystkich możliwych akcji. Postąpiono zgodnie ze wskazówkami z podrozdziału 2.3. Dodajemy niezbędne krawędzie wynikające z warunków, jak i efektów każdej z akcji oraz mutexy (oznaczone przerywanymi, niebieskimi liniami).



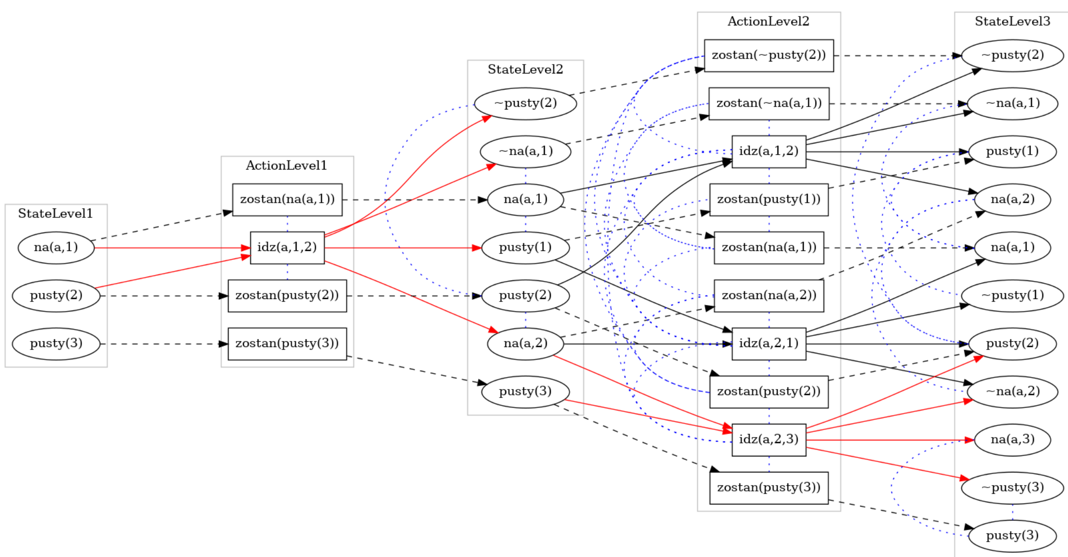
Rysunek 2.16: Wygenerowany drugi stan grafu bezpośrednio wynikający z pierwszego poziomu akcji

Przy pomocy efektów akcji automatycznie wykreowano następny poziom stanów. Zgodnie z schematem planowania należy sprawdzić, czy oczekiwany cel $na(A,3)$ znajduje się w zbiorze stanów poziomu drugiego. Po szybkiej analizie okazuje się, iż cel nie znajduje się na wskazanym poziomie, więc należy dokonać rozszerzenia grafu planującego o kolejny poziom. Następuje to w sposób analogiczny.



Rysunek 2.17: Przykład kolejnej iteracji algorytmu

Po wygenerowaniu kolejnej warstwy algorytmu oraz zaznaczeniu wszystkich mutexów zauważono, iż stan $na(A,3)$ znajduje się w zbiorze stanów **StateLevel3**. Następnym krokiem jest sprawdzenie, czy taki zbiór akcji, który doprowadzi nas do stanu $i-1$ bez wprowadzania żadnych sprzeczności. Okazuje się, iż istnieje taka para akcji: $idz(A,2,3) orazzostan(pusty(1))$. Zgodnie z schematem generowania planu należy zejść iteracyjnie aż do warunków początkowych aby uzyskać poprawny plan. W omawiany przykładzie łatwo zauważyć, iż przy pomocy akcji $idz(A,1,2)$ oraz $pusty(3)$ udało się z powrotem otrzymać stan początkowy. Ze względu na fakt, iż człowiek jest w stanie sprawnie wydedukować akcje podtrzymujące na podstawie akcji aktywny dla każdego poziomu, przy graficznym przedstawieniu planu (w formie opisowej i graficznej) z reguły akcje podtrzymujące będą pomijane.



Rysunek 2.18: Graf planujący z zaznaczonymi akcjami aktywnymi prowadzącymi do uzyskania wskazanego celu

Powyższy rysunek przy pomocy czerwonych strzałek przedstawia akcje aktywną, wraz z jej bezpośrednimi warunkami zajścia oraz efektami, które prowadzą do uzyskania planu. Zgodnie z intuicją, najbardziej optymalny plan przeniesienia robota A z kafelka 1 na kafelek 3 to:

$$[[idz(A,1,2)],[idz(A,2,3)]] \tag{2.11}$$

Ze względu na fakt, iż na danym poziomie może odbywać się więcej niż jedna akcja, każdy krok algorytmu oznaczany jest za pomocą **listy**, którą możemy utożsamić z matematycznym zbiorem.

W ten oto sposób wygenerowano poprawny plan stosując algorytm **GRAPHPLAN**.

2.8 Własności GRAPHPLAN'u

Działanie GRAPHPLAN'u można przedstawić również jako swego rodzaju iteratywne przeszukiwane wszerz. Dla każdego poziomu generowane są wszystkie możliwe stany, by następnie móc sprawdzić, czy cele do osiągnięcia znajdują się w utworzonym zbiorze. Jeśli rzeczone cele nie znajdują się, algorytm wykonuje kolejną iterację, poszerzając swoją wiedzę o świecie. Powyższe przeszukiwanie wykonuje się aż do momemntu, gdy uda się utworzyć satysfakcjonujący plan Dzięki tej własności łatwo pokazać, iż:

Twierdzenie 2.1 *Każdy wyprodukowany plan przez algorytm GRAPHPLAN jest planem legalnym, czyli spełnialnym dla zadanych warunków początkowych, akcji oraz celów. Ponadto, jeśli dla opisanego świata istnieje plan GRAPHPLAN zawsze go znajdzie.*

oraz

Twierdzenie 2.2 *Algorytm zawsze zwraca najkrótszy plan, czyli plan optymalny.*

Dowód. W pierwszym kroku algorytm generuje nowy poziom stanów. Następnie sprawdza, czy cele znajdują się wśród stanów. Jeśli odpowiedź jest twierdząca, dobiera zbiór akcji, dzięki którym udało się spełnić postawione wymagania. Jeśli odpowiedź jest przecząca, algorytm generuje kolejną warstwę. Niech warstwa i będzie warstwa, w której nie znajdują się wszystkie stany ze zbioru determinowanego przez zdefiniowane cele. W tym przpyadku dla $i+1$ warstwy algorytm próbuje wygenerować plan cofając się do stanu i. Jeśli uda się odpowiednio dobrać akcje cofa się jeszcze dalej aż do stanu początkowego i kończy działanie, jako wynik zwracając plan. Powyższy opis determinuje optymalność planu. ♦

Ponadto należy się również przyjrzeć mechanizmom kreowania kolejnych stanów dla poszczególnych warstw. Algorytm robi to przyrostowo. Wszystkie istniejące już stane przenoszone są przy pomocy akcji podtrzymujących do następnej warstwy, natomiast akcje aktywne produkują nowe efekty- dodające bądź usuwające. Mimo iż akcja z perspektywy człowieka jest **usuwająca**, z perspektywy komputera jest kolejną informacja, która również zostaje dodana do zbioru stanów.

Korzystając z powyższej własności sformułowano następujący wniosek:

Twierdzenie 2.3 *Liczba stanów w warstwie **i+1** jest zawsze większa bądź równa liczbie stanów w warstwie **i***

Skutki ów twierdzenia są dobrze uwidocznione w przykładach grafów planujących takich jak 2.19. Twierdzenie 2.3 generuje ciekawy wniosek. Niech oznaczenie $WSL(i)$ oznacza liczbę stanów wchodzą w skład i-tej warstwy.

Lemat 2.1 $\exists k \in \mathbb{N}\ \forall i > k, i \in \mathbb{N}$ *: WSL(i+1) = WSL(i)*

Co sprowadza się do tego, iż od pewnego momentu liczba stanów dla każdej kolejnej warstwy jest stała.

Dowód. Zgodnie z 2.3 wiadomym jest, iż zawsze spełniona jest zależność $WSL(i+1) > WSL(i)$. Należy pokazać, iz od pewnego momentu liczby stanów w warstwach dążą do stałej liczby. Ze względu na strukturę języka STRIPS istnieje skończona liczba akcji. W ów modelu nie istnieją akcje niedeterminstyczne, bądź jakiekolwiek niespodziewane efekty uboczne przeprowadzonych akcji. Algorytm również rozpoczyna pracę z góry znaną liczbą obiektów, a każdy z obiektów może znajdować się w skończonej liczbie stanów takich jak *na* czy *pusty*. Algorytm rozszerza kolejne poziomy stanów przy pomocy akcji aktywnych, jednakże na podstawie skończonej liczby obiektów oraz skończonych stanów, w jakich ów obiekty mogą się znajdować algorytm, dla każdego z obiektów, może wywnioskować jedynie skończoną liczbę akcji, które może wykonać. Ostatecznie sprowadza się to do sytuacji, w której ze względu na skończoność wszystkich określonych dziedzin, i-ty poziom stanów algorytmu zawiera wszystkie możliwe stany dla wszystkich możliwych obiektów. Od tego momentu mimo przeprowadzania kolejnych akcji podtrzymujących bądź aktywnych nie powstają już kolejne stany, gdyż zbiór efektów każdej z akcji ma już swoje odwzorowanie w poprzedniej warstwie. Stąd wniosek, iż istnieje moment krytyczny, dla którego dochodzi do stabilizacji liczby stanów w danej warstwie. ♦

Definicja 2.9 ***Spłaszczeniem*** *algorytmu jest sytuacja, w której wygenerowanie kolejnego stanu nie prowadzi do uzyskania nowych informacji o świecie.*

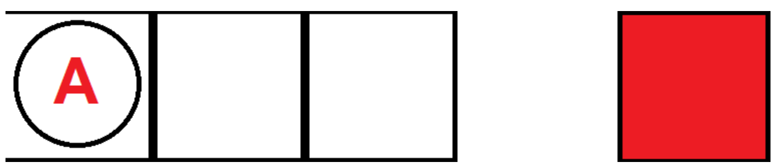
Powyższy lemat jest kluczowym w kontekście rozwiązywania tak zwanego *Problemu stopu* dla algorytmu GRAPHPLAN.

2.8.1 Złożoność obliczeniowa

Twierdzenie 2.4 *Niech problem do rozwiązania przy pomocy planu składa się z n obiektów, p stanów w warstwie początkowej, m operatorów przedstawionych zgodnie z zasadami języka STRIPS, posiadających stałą liczbę parametrów. Dodatkowo, niech l będzie długością najdłuższej listy efektów dodających dla zbioru akcji. Wtedy rozmiar t poziomowego grafu planującego wygenerowanego przez algorytm, jak i czas jego generacji jest wielomianowy w zależności od zmiennych n, m, p, l i t*

Co jest znacznym usprawnieniem względem poprzednich metod. Dowód tego twierdzenia można znaleźć w oryginalnej pracy autorów [2].

2.8.2 Problem stopu



Rysunek 2.19: Przykład świata, w którym osiągnięcie celu, którym jest kafelek oznaczony kolorem takim jak kolor użytej czcionki do oznaczenia sygnatury robota, jest niemożliwe ze względu na absencję jednego z kafelków, będących łącznikiem, między robotem a jego celem.

Wcześniejsze przykłady rozpatrywały jedynie sytuację, w której algorytm zawsze znajdował plan, gdyż przedstawione światy były spreparowane w taki sposób, aby rzeczony plan zawsze istniał. Jednakże nie zawsze musi tak być, więc należy zabezpieczyć algorytm również i przed taką ewentualnością. Łatwo zauważyć, iż obecna struktura planu nie została utworzona z myslą o takiej sytuacji- algorytm będzie generował coraz to nowsze warstwy bezskutecznie próbując uzyskać plan dla zdefiniowanego celu. Ostatecznie dojdzie do całkowitego zapętlenia pracy algorytmu. Z tego powodu pierwszym pomysłem na uzbrojenie algorytm w mechanizm detekcji nieskończonej pętli jest sprawdzenie rozmiaru zbioru stanów dla dwóch następujących po sobie warstw. Jeśli warstwa nowsza składa się z większej liczby stanów, algorytm uzyskuje nowe informacje o świecie i jest przygotowany do dalszego trawersowania po grafie planującym, jednakże gdy ów liczby są sobie równe dochodzi do swego rodzaju stagnacji- wszystkie możliwe ruchy nie wprowadzają nowych informacji o świecie. Jeśli dla poprzedniego stanu nie udało się utworzyć planu, algorytm wnioskuje, iż również dla tego, jak i przyszłych, utworzenie stanu będzie niemożliwe, co powoduje zwrócenie przez program informacji o braku możliwości utworzenia odpowiedniego planu i jego terminację.

Po wprowadzeniu powyższej modyfikacji możliwym jest wyciągnąć następujący wniosek

Wniosek 2.1 *GRAPHPLAN zawsze kończy swoje działanie.*

a co za tym idzie

Wniosek 2.2 *Odpowiedzi GRAPHPLANU dla zadanych danych wejściowych przyjmują jedynie dwie formy:*

1. *plan, którego realizacja prowadzi do uzyskania wskazanych celów*
2. *informacja o tym, iż nie istnieje plan spełniający oczekiwania użytkownika*

Powoduje to, iż GRAPHPLAN jako algorytm jest kompletny i zawsze zakończy swoje działanie w skończonym czasie. Jest to własność, której większość z planerów korzystających z częściowego porządku nie posiada w swoim arsenale.

Rozdział 3

Programowanie ograniczeń

3.1 Wprowadzenie

Programowanie ograniczeń (inna nazwa: technologia więzów) jest narzędziem wykorzystywanym do rozwiązywania problemów z dziedzin kombinatoryki, sztucznej inteligencji, czy planowania jak i harmonogramowania zadań. W skład tego podejścia do programowania często wyróżnia się dwa elementy: ograniczenia (zwane również stałymi) oraz problem rozwiązywania ograniczeń (ang. Constraint Satisfaction Problem, CSP) Poniżej dokonano formalnego zdefiniowania powyższych komponentów:

Definicja 3.1 ***Problem rozwiązywania ograniczeń, CSP*** *jest następującą trójką:*

$$CSP = (V,D,C) \tag{3.1}$$

gdzie:
$V = \{x_1,x_2,...,x_n\}$ *oznacza zbiór zmiennych wykorzystywanych do opisu problemu*
$D = \{d_1,d_2,...,d_n\}$ *oznacza zbiór dziedzin wyżej wspominanych zmiennych. W ramach rozważań zawartych w rzeczonej pracy rozpatrywane będą takie* d_i*, które są zbiorami zawierającymi skończoną liczbę potencjalnych wartości zmiennej* x_i*.*
$C = \{c_1,c_2,...,c_m\}$ *oznacza zbiór ograniczeń*

Definicja 3.2 ***Ograniczenami*** *(inne nazwy: stałe, więzy) nazywamy zmienne oraz zależności między nimi, które muszą zostać spełnione w ramach rozwiązywania problemu ograniczeń. Określamy je przy pomocy pary:*

$$C = (S,R) \tag{3.2}$$

gdzie:
S jest krotką wszystkich zmiennych wchodzących w skład relacji
R jest relacją, która definiuje jakie wartości mogą przyjąc zmienne, które w niej uczestniczą.

Relacje często przedstawia się przy pomocy zbioru zawierającego krotki, które składają się ze wszystkich przyporządkowań wartości do odpowiednich zmiennych. Przy następujących definicjach oczekiwanym celem będzie utworzenie mechanizmu rozwiązującego zadany problem. Jego wynikiem będzie zbiór wszystkich zmiennych, oraz krotek, które będą zawierały odpowiednie wartości przyporządkowane dla zmiennych.

Definicja 3.3 ***Krotka*** *(ang. tuple) - struktura danych, która w systemach informatycznych odzwierciedla uporządkowany ciąg wartości*

Dodatkowo wprowadza się termin **arności** ograniczenia:

Definicja 3.4 ***Arność ograniczenia*** *(ang. arity) związana jest z liczbą unikalnych zmiennych, która w nią wchodzi.*

Najpopularniejszymi typami ograniczeń są:

1. Ograniczenia o arności 1, zwane ograniczeniami **unarnymi** (w tym przypadku z reguły będą one ściśle związane z dziedziną, raczej nie będą występowały w kontekście ograniczeń)

2. Ograniczenia o arności 2, zwane ograniczeniami **binarnymi**

3. Ograniczenia o arności 3, zwane ograniczeniami **ternarnymi**

Tak jak ograniczenie może mieć swoją arność, tak dla CSP również zdefiniowano pojęcie arności w lekko zmodyfikowany sposób

Definicja 3.5 ***Arność*** *CSP o wartości i zawiera w sobie wszystkie typy ograniczeń od arności 1 aż do arności i*

Wedle powyższego binarne CSP zawiera w sobie jedynie ograniczenia unarne jak i binarne.

Przykład 3.1 *Niech będzie dane równanie* $x + y = z$*, gdzie* $x,y,z \in \{0,1\}$*. Łatwo zauważyć, iż zadane równanie jest automatycznie ograniczeniem wpływającym na prezentowane zmienne. Przyporządkowując odpowiednie wartości do zbiorów z definicji otrzymano*

1. $V = \{x,y,z\}$

2. $D = \{d_x,d_y,d_z\}$*, gdzie* $d_x,d_y,d_z = \{0,1\}$

3. $C = \{x + y = z\}$

Arność *ograniczenia występujące w przedstawionym przykładzie wynosi 3, determinowane jest to liczbą zmiennych, która wchodzi w jej skład. Rozwiązaniem tego problemu będą następujące dopasowania:*
$((x,y,z),\{0,0,0\},\{1,0,1\},\{0,1,1\})$

Odnalezienie rozwiązań z przykładu 3.1 było trywialne ze względu na małą liczbę zmiennych, wąskie dziedziny oraz tylko jedno wprowadzone ograniczenie. Podobnie jak z planowaniem, wprowadzenie dodatkowych zmiennych, powiększanie dziedzin oraz zbioru ograniczeń znacznie wpływa na skomplikowanie odnajdowania rozwiązania.

3.2 Pojęcie ustalenia i spójności

W ostatecznej formie problem ograniczeń wyszukuje rozwiązanie przy pomocy ustalenia wartości zmiennych, jednakże naistotniejsza jest droga, jaką pokonuje, aby ów ustalenia uzyskać. W tej sekcji należy wprowadzić kilka dodatkowych definicji:

Definicja 3.6 *Ustalenie zmiennych jest* ***spójne****, gdy nie jest w konflikcie z żadnym z ograniczeń.*

Ustalenie spójne również w nomenlakturze programowania ograniczeń nazywane jest ustaleniem **legalnym**. Nadanie zmiennym X oraz Y wartość 1 w przykładzie 3.1 prowadzi do konfliktu, gdyż nie istnieje wartość 2 w zbiorze D_z.

Definicja 3.7 ***Kompletnym ustaleniem*** *nazywamy takie ustalenie, w którym wszystkie zmienne posiadają ustaloną wartość.*

Łatwo zauważyć, iż kompletne ustawienie jest jednocześnie rozwiązaniem problemu ograniczeń. Z tego płynie następujący wniosek:

Wniosek 3.1 *Każde rozwiązanie problemu ograniczeń jest spójne.*

Dodatkowo w źródłach [7] definiuje się częściowe ustalenie oraz częściowe rozwiązanie. Zgodnie z nazewnictwem częściowe ustalenie związane jest z sytuacją, gdy jeszcze nie wszystkie zmienne mają dokładnie określone wartości, natomiast częściowe rozwiązanie w praktyce identyfikuje się jako spójne częściowe ustalenie.

Po wprowadzeniu powyższych definicji należy rozpocząć rozważania na temat tego, w jaki sposób problem ograniczeń może zostać rozwiązany. Pierwszym z podejść może być ustalenie wartości dla zmiennych poprzez analizę ograniczeń, jakie między nimi występują. Ten proces nazywany jest **propagacją**

ograniczenia. Dzięki podstawowej analizie ograniczeń algorytm może wyeliminować wartości nadmiarowe znajdujące się w zadanych dziedzinach, co znacząco wpłynie na przyszłościowe osiągi pod kątem czasowym. Często propagacja ograniczeń jest wykonywana jako *preprocessing step*, czyli jako krok, który zostanie wykonany jeszcze przed rozpoczęciem prawdziwej pracy nad problem.

Przy rozwiązywaniu problemów związanych z ograniczeniami bardzo intuicyjną reprezentacją, jest reprezentacja w formie *grafu*. Poprzez wierzchołki oznacza się zmienne wchodzące w skład problemu, natomiast poprzez krawędzie- binarne ograniczenia, która między nimi występują. Z tego powodu pożądanym zabiegiem będzie przedstawienie wszystkich ograniczeń n-arnych w formie binarnej.

Kluczem do uzyskania poprawnego efektu propagacji ograniczeń jest skorzystanie z pojęcia zdefiniowanego jako **lokalna spójność**. Istnieją różne typy lokalnej spójności:

- **Spójność wierzchołkowa** (ang. node consistency) - Zmienna jest wierzchołkowo spójna, gdy wszystkie wartości znajdujące się w jej dziedzine spełniają zdefiniowane przez nią ograniczenie unarne. Graf jest wierzchołkowo spójny, gdy wszystkie wierzchołki wchodzące w skład grafu są wierzchołkowo spójne. Zachowanie wierzchołkowej spójności z reguły sprowadza się do **zawężania** dziedziny zmiennej.

- **Spójność krawędziowa** (ang. arc consistency/edge consistency)- Zmienna jest spójna krawędziowo wtedy, gdy każda wartość z jej dziedziny spełnia binarne ograniczenia zmiennej. Graf jest krawędziowo spójny gdy każda para zmiennych jest ze sobą krawędziowo spójna.

- **Spójność ścieżki** (ang. path consistency) - Dwie zmienne a,b są spójne w kontekście ścieżki z trzecią zmienną c, gdy każde przypisanie wartości do zmiennych a,b spełniające ograniczenie występujące między rzeczonymi zmiennymi dodatkowo spełnia ograniczenie między zmiennymi a,c oraz c,b.

- **K-spójność**- CSP jest k-spójne, gdy dla każdego spójnego ustalenia zawierającego $k-1$ zmiennych można dołączyć k-tą zmienną bez załamiania spóności. Pojęcie 2-spójność jest tożsame z spójnością krawędziową, a 3-spójność- ze spójnością ścieżki.

Sprowadzanie problemu ograniczeń do sytuacji, w której zachodzi lokalna spójność jest określane mianem **propagacją ograniczeń**. Jest to o tyle istotne, iż jedną z głównych metod rozwiązywania problemu ograniczeń jest sprowadzenie dziedzin zmiennych do pojedynczej wartości- wtedy całkowitym ustaleniem jest nadanie zmiennej jedynej wartości w swojej dziedzinie. Propagacja ograniczeń jest silnym mechanizmem wykrywającym niespójności- jeśli przy próbie propagacji któraś z wartości zostałaby z pustą dziedziną, wtedy należałoby przerwać rozwiązywanie problemu wraz ze zwróceniem informacji o fakcie, iż nie ma takiego ustalenia zmiennych przy obecnych dziedzinach, dla którego rozpatrywane ograniczenie jest rozwiązywalne.

3.3 Ograniczenia globalne

3.4 Wyszukiwanie rozwiązań

Podstawowa metoda rozwiązywania problem z dziedziny ograniczeń nosi miano metody **cofającej** (ang. backtracking). Działa ona zbliżenie do mechanizmu przeszukiwania grafu wgłąb. Na początku wybierana jest jedna ze zmiennych. Dla każdej wartości z dziedziny dochodzi do częściowego ustalenia- zmienna otrzymuje wartość równą pierwszej wartości w swojej dziedzinie. Następnie, korzystając z tej informacji dochodzi do propagacji ograniczeń. Jeśli wybrana została poprawna wartość propagacja ograniczeń doprowadzi do znalezienia rozwiązania problemu. Istnieje również sytuacja, w której wybrana wartość prowadzi do slepego zaułka, czyli do sytuacji, w której nie istnieje odpowiednie ustalenie zmiennych. Wtedy należy **cofnąć** się do miejsca, w którym zmiennej nadaliśmy wartość i spróbować innego ustalenia. Połączenie mechanizmu cofnięcia wraz z propagacją ograniczeń, czyli z utrzymywaniem lokalnej spójności na każdym etapie wyszukiwania rozwiązania jest znakomitą techniką poprawiającą wydajność. Dzięki propagacji ograniczeń często dochodzi do sytuacji, w której wyżej wymienione ślepe zaułki są eliminowane zanim mechanizm cofania je rozpatrzy.

Drugą metodą jest metoda nazywana **wyszukiwaniem lokalnym** (ang. local search). Metoda cofająca, jak sama nazwa wskazuje, próbowała dokonać częściowego ustalenia, by następnie przy propagacji ograniczeń udowodnić, iż rozpatrywane częściowe ustalenie jest prawidłowe i generuje odpowiednie całkowite ustalenie. Metodologia wyszukiwania lokalnego różni się w swojej filozofii tym, iż na samym początku dochodzi do pełnego ustalenia zmiennych. W znacznej większości przypadków ów pełne ustalenie jest nieprawidłowe, to znaczy nie spełnia wszystkich ograniczeń. Wtedy mechanizm próbuje na bieżąco naprawiać sytuację modyfikując całkowite ustalenie w takich sposób, aby ustalenie nadal było całkowite jednocześnie spełniając ograniczenie, które wcześniej powodowało konflikty. Jeśli algorytm będzie w stanie rozwiązać wszystkie ograniczenia uzyska odpowiednie ustalenie zmiennych.

Porównując powyższe dwie metody łatwo zauważyć, iż różnią się one od siebie w znacznym stopniu, nie tylko w samej filozofii działania, lecz także w efektach.

Definicja 3.8 ***Algorytmem kompletnym*** *jest algorytm, który gwarantuje uzyskanie rozwiązania oraz jest w stanie wykryć, gdy takowe rozwiązanie nie istnieje*

Przeciwieństwem algorytmu kompletnego jest algorytm **niekompletny**, czyli taki, który nie gwarantuje uzyskania optymalnego rozwiązania oraz wykrycia, czy problem posiada rozwiązanie. Algorytmy niekompletne natomiast są o wiele szybsze oraz dobrze przybliżają optymalne rozwiązanie.

Zgodnie z powyższym metoda cofająca jest przykładem algorytmu kompletnego- systematycznie generuje nowe ustalenia oraz propagacje ograniczeń, natomiast wyszukiwanie lokalne jest przykładem algorytmu niekompletnego- łatwo sobie wyobrazić sytuację, iż naprawienie jednego ograniczenia może nieustannie generować zepsucie kolejnego. W trakcie wyboru algorytmu ważnym jest, aby znać jego zalety jak i wady. Do implementacji GRAPHPLANU został wykorzystany mechanizm cofania, aby zwracany przez algorytm plan był zawsze optymalny.

Powyżej wymienione metody posiadają wiele dodatkowych usprawnień oraz specjalnie zdefiniowanych heurystyk, z którymi czytelnik może zapoznać się kierując się do następującej pozycji w bibliografii [1]

3.5 Programowanie w logice a ograniczenia

3.6 Obrazowe przykłady

//SEND + MORE = MONEY, N HETMANÓW, czy problem plecakowy? //Wszystkie trzy będą pewnie zbędne

3.7 Wykorzystanie w algorytmie

Podczas graficznego prezentowania przykładów programowania ograniczeń często wykorzystywaną strukturą był graf, chociażby w sekcji omawiającej pojęcie lokalnych spójności (3.2). Ze względu na ów powiązanie między programowaniem ograniczeń a GRAPHPLAN'em do podstawowego opisu GRAPHPLANU z rodziału 2 2 dodano funkcjonalności opisane w powyższych rozdziałach.

Każdy ze stanów oraz akcji zawiera w sobie dodatkowy **indykator**. Jest to liczba ze zbioru liczb całkowitych, o której należy mysleć bardziej w kontekście wartości bool'owskich $\{prawda, fasz\}$. Wartość liczby równa 0 indikuje fałszywość stanu, natomiast wartość większa od 0 indukuje jego prawdziwość. Przy pomocy indykatorów program ustala, które stany, bądź akcje są w danej warstwie prawdziwe, czyli występują w świecie oraz takie, które w ów świecie w danym momencie nie występują, czyli są fałszywe. Odbywa się to w następujący sposób:

1. Wszystkie stany wchodzące w stan początkowy otrzymują indykator równy 1, gdyż są aktualnie prawdziwe w rozpatrywanym świecie.

2. W trakcie generowania akcji następuje utworzenie powiązania między warunkami zajścia, akcjami oraz ich efektami. Akcja zostaje powiązane ze swoim warunkiem następującym ograniczeniem: wartośc indykatora akcji jest mniejsza bądź równa wartości indykatora warunku. Należy to rozumieć w następujący sposób- jeśli warunek jest prawdziwy to akcja **może** zachodzić w świecie, natomiast jeśli warunek jest nieprawdziwy, czyli ma indykator równy 0, akcja automatycznie dostaje indykator równy 0. Efekt zostaje powiązany ze swoją akcją poprzez następujące ograniczenie: jeśli jakakolwiek akcja w danej warstwie, która ma dany stan za efekt zachodzi, wtedy również i efekt w nim

występuje. Jeśli wszystkie akcje generujące ów efekt mają indykator równy 0 wtedy efekt nie może zachodzić na danym poziomie, więc również otrzymuje indykator równy 0.

3. Dochodzi do sprawdzenia relacji wzajemnego wykluczania poprzez sprawdzenie indykatorów dwóch stanów- Jeśli ich iloczyn jest równy 0, wtedy dwa stany nie mogą razem występować na danym poziomie.

4. Przy dokładnej realizacji kroków 2 i 3 algorytm jest w stanie wygenerować kolejny poziom stanów. Po wygenerowaniu dochodzi do sprawdzenia, czy wszystkie stany zawarte w zbiorze przechowującym cele mają indykatory równe 1. Jeśli nie, rzeczony proces jest powtarzany aż do otrzymania pożądanego skutku.

5. Gdy wszystkie cele otrzymają indykator 1, program przelicza wszystkie indykatory, dzięki czemu jest w stanie bezbłędnie określić, który stan bądź która akcja na danym etapie przetwarzania świata znajdują się w nim bądź nie. Ów mechanizm jest silnie wykorzystywany przy generowaniu grafów, przedstawiających zachodzące w świecie zmiany.

Dokładne sformuowanie ograniczeń przy pomocy symboli matematycznych odbędzie się w sekcji poświęconej implementacji algorytmu.

Rozdział 4

Implementacja

4.1 Połączenia między komponentami

//DO DOKŁADNEGO ZREDAGOWANIA W OSTATNI WEEKEND LISTOPADA PO WPROWADZENIU POPRAWEK W WARSTWIE GRAFICZNEJ

4.2 Implementacja algorytmu

4.3 Generowanie grafów

4.4 Interfejs użytkownika

Rozdział 5

Instalacja i wdrożenie

5.1 Instalacja pakietu SWI-Prolog

//DO DOKŁADNEGO ZREDAGOWANIA W OSTATNI WEEKEND LISTOPADA PO WPROWADZENIU POPRAWEK W WARSTWIE GRAFICZNEJ

5.2 Instalacja języka Python

5.3 Sposób uruchomienia programu

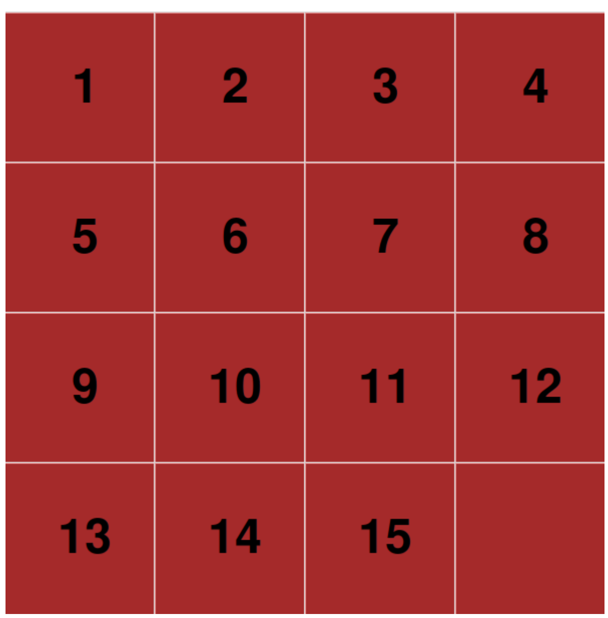
Rozdział 6

Testy

6.1 15

6.1.1 Wprowadzenie

Piętnastka (fr. taquin), znana również w Polsce o nazwie *przesuwanka*, w zapisie często podawana przy pomocy liczbowego odpowiednika (15) jest grą w formie specyficznej układanki, której powstanie datuje się na koniec XIX wieku. Składa się ona z 15 klocków oraz ramki, pierwotnie drewnianej. Ramka zaprojektowana została specjalnie z myślą o pozostawieniu jednego wolnego miejsca, aby móc w łatwy sposób przesuwać klocki sąsiadujące z miejscem pustym. Celem gry jest ułożenie klocków w określony sposób, najcześciej w porządku rosnącym czytając od lewej do prawej rzędami, z określonego stanu początkowego. Częstym zabiegiem stosowanym przez twórców ów układanki jest konwersja liczb na części obrazka, aby zachęcić do gry młodszych odbiorców.



Rysunek 6.1: Wygląd układanki piętnastki wygenerowany przy pomocy zaimplementowanej w ramach pracy warstwy graficznej

14	3	4	2
1	10	5	13
11	6	9	7
8		15	12

Rysunek 6.2: Losowa rozwiązywalna permutacja układanki

Rysunek 6.3: Piętnastka w formie obrazkowej. Źródło: http://mypuzzlecollection.blogspot.com/2012/08/mc-escher-birds-fish-and-turtles.html

6.1.2 Teoria

W 1878 roku amerykański wynalazca gier i zagadek (między innymi zagadek szachowych) **Samuel Loyd** ze względu na swój fach nie przeszedł obojętnie koło piętnastki proponując ułożenie układu rosnącego z pozycji, która od początkowej różniła się pozycjami jedynie dwóch klocków numerowanych odpowiednio 14 i 15 [3]. Problem ów stał się na tyle popularny, iż została wyznaczona nagroda 1000 dolarów dla osoby, której udałoby się znaleźć prawidłowe rozwiązanie przygotowanego przez Pana Samuela problemu.

Niemożliwość ułożenia problemu przez bardzo długi czas doprowadziła do pierwszych poważniejszych rozważań nad z pozoru trywialną łamigówką. Efektem prac matematyków było parę zaskakujących wniosków, które ostatecznie doprowadziły do udowodnienia, iż wyżej przedstawiona łamigłówka jest nierozwiązywalna.

Lemat 6.1 *Nie wszystkie ustawienia początkowe piętnastki są możliwe do rozwiązania.[5]*

Wynika to z faktu, iż rozwiązywalne są jedynie rozwiązanie oo parzystej liczbie inwersji. Zagadka Pana Lyod'a jest ustawieniem nieparzystym jeśli chodzi o inwersje. Prowadzi to do następującego wniosku:

Wniosek 6.1 *Istnieje* $\frac{16!}{2} = 10461394944000$ *rozwiązywalnych ustawień.*

Dodatkową ciekawostką istotną z perspektywy wykonywanych testów jest minimalna liczba posunięć, którą należy wykonać, aby z rozwiązywalnego stanu osiągnąć wcześniej wyznaczony cel. Mianem **boskiej liczby** w odniesieniu do przesuwanki określą się największą liczbę posunięć, którą trzeba osiągnąć, aby rozwiązać najtrudniejsze ułożeniem początkowe. Przy pomocy matematyki naukowcy odnaleźli najtrudniejsze ustawienia oraz obliczyli ów liczbę, co zaprezentowano w następującym lemacie:

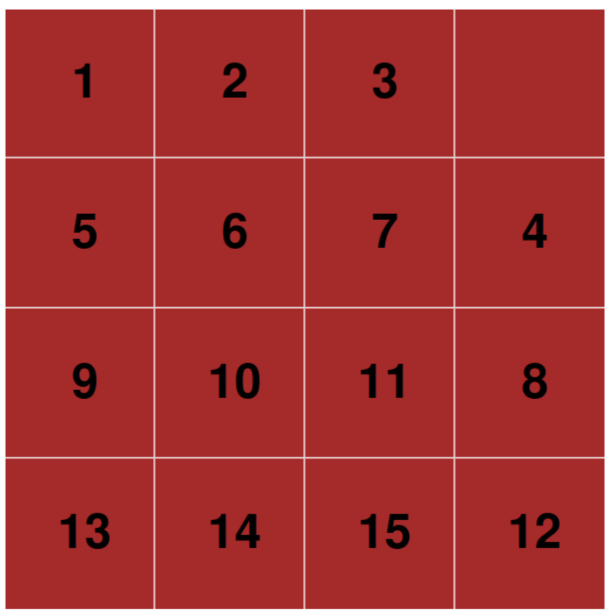
Lemat 6.2 ***Boska liczba*** *dla 15-elementowej przesuwanki wynosi* ***80****.* [4]

Oznacza to, iż maksymalna liczba kroków algorytmu w żadnym wypadku nie powinna przekroczyć liczby 80.

W trakcie poniżej opisanego testu sprawdzono plany przesuwania odpowiednich klocków, aby w jak najmniejszej możliwej liczbie ruchów otrzymać odpowiedni stan końcowy, przy okazji sprawdzono osiągnięcia czasowe jak i porównano otrzymane wyniki z popularnymi herustykiami spersonalizowanymi pod rozwiązywanie ów układanki.

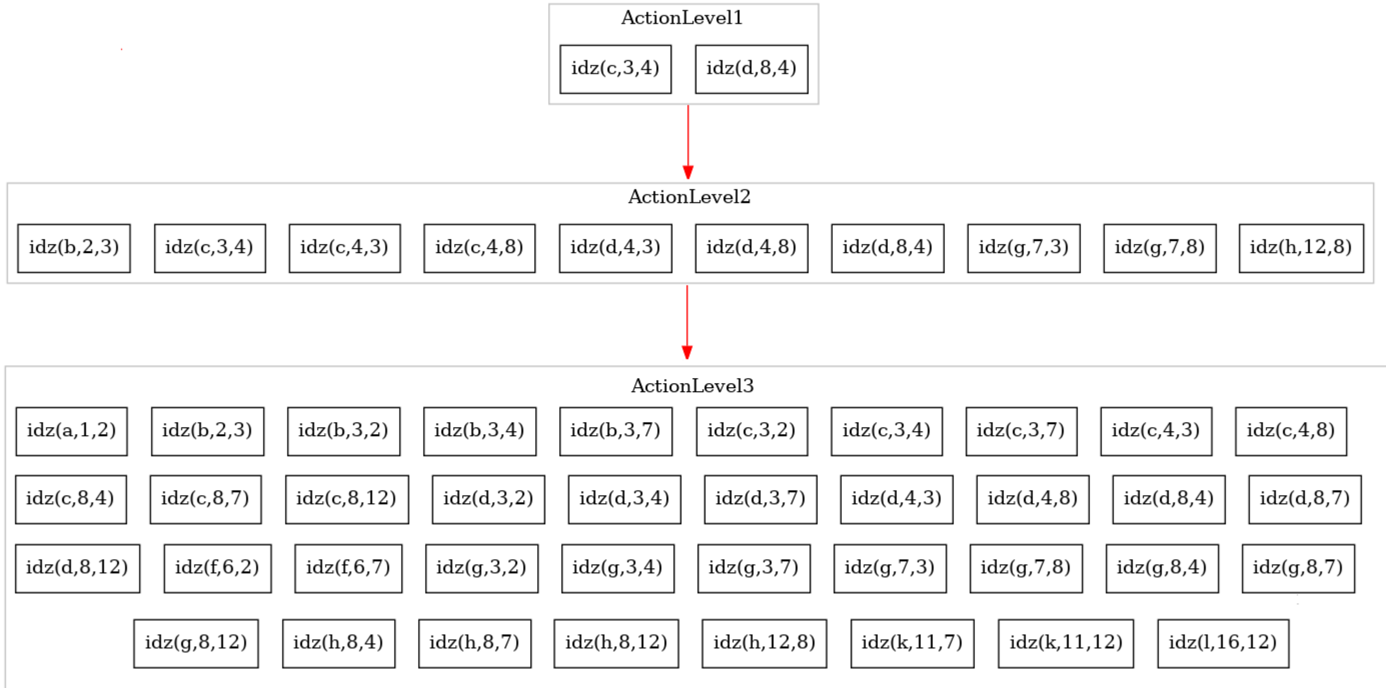
6.1.3 Przykład

Na podstawie następującego przykładu zostanie przedstawiony schemat rozwiązywania układanki przez algorytm:

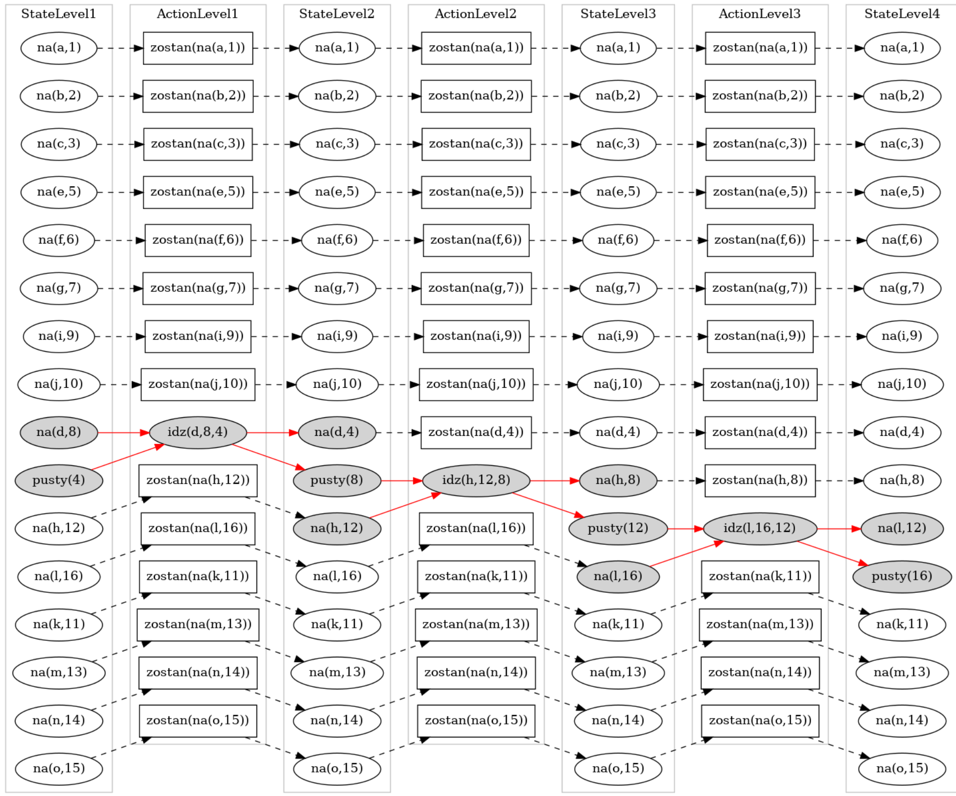


Rysunek 6.4: Przykładowe startowe ułożenie przesuwanki

Algorytm buduje graf na podstawie jedynej zdefiniowanej w problemie akcji: przesuwania klocka. Robi to aż do momentu, gdy kafelki nie będą ułożone w wczesniej zdefiniowanej kolejności. Dla zdefiniowanego powyżej przykładu w pierwszym kroku algorytm ma jedynie dwie możliwości akcji aktywnych: zamiana klocka pustego z klockiem o numerze 3 lub klockiem o numerze 4. Na tej podstawie generuje kolejny poziom stanów. Dla załączonego przykładu optymalnym rozwiązaniem jest odpowiednio zamiana pustego kafelka z kafelkami: 4,8,12, co zostało poprawnie wyznaczone przez GRAPHPLAN. Poniżej przedstawiono zbiory akcji przeanalizowane przez algorytm w danym kroku jak i uproszczony graf planujący.



Rysunek 6.5: Akcje rozpatrywane przez algorytm w danym kroku



Rysunek 6.6: Uproszczony graf planujący wygenerowany przez algorytm GRAPHPLAN przedstawiający stan każdego kafelka w danej warstwie. Węzły wypełnione kolorem szarym obrazują stany, które są warunkami zajścia jak i efektami wykonywanej w danej warstwie akcji

6.1.4 Szczegóły implementacyjne

Ważnym jest, aby przedstawić omawiany świat zgodnie z wytycznymi ustalonymi przez język **STRIPS**. Z tego powodu należy dokładnie okreslić każdą z istniejących w świecie relacji oraz poprawnie określić cel, jak i warunki początkowe.

6.1.5 Wyniki

UWAGA: Testy czasowe zaprezentowane w poniższych tabelach tyczą się osiągów samego algorytmu. Oznacza to, iż na czas wykonywania prób wyłączone zostały wszystkie poboczne funkcjonalności takie jak generowanie grafu, czy prezentowanie rozwiązania w formie graficznej. Wykonanie poniższych badań w aplikacji może skutkować innymi wynikami, zwykle dłuższymi. Należy mieć to na uwadze przy potencjalnej próbie odtwarzania badań.

6.1.6 Młodsza siostra- ósemka

6.1.7 Wyniki dla 8

6.1.8 Wnioski

6.2 CargoBot

6.2.1 Wprowadzenie

6.2.2 Przykład

6.2.3 Szczegóły implementacyjne

6.2.4 Wyniki

6.2.5 Wnioski

6.3 Przemieszczanie w przestrzeni

6.3.1 Wprowadzenie

6.3.2 Przykład

6.3.3 Szczegóły implementacyjne

6.3.4 Wyniki

6.3.5 Wnioski

6.4 Wieża Hanoi

6.4.1 Wprowadzenie

6.4.2 Przykład

6.4.3 Szczegóły implementacyjne

6.4.4 Wyniki

6.4.5 Wnioski

6.5 Problem komiwojażera

6.5.1 Wprowadzenie

6.5.2 Przykład

6.5.3 Szczegóły implementacyjne

6.5.4 Wyniki

6.5.5 Wnioski

6.6 Osiem Hetmanów

6.6.1 Wprowadzenie

6.6.2 Przykład

6.6.3 Szczegóły implementacyjne

6.6.4 Wyniki

Podsumowanie

W podsumowaniu należy określić stan zakończonych prac projektowych i implementacyjnych. Zaznaczyć, które z zakładanych funkcjonalności systemu udało się zrealizować. Omówić aspekty pielęgnacji systemu w środowisku wdrożeniowym. Wskazać dalsze możliwe kierunki rozwoju systemu, np. dodawanie nowych komponentów realizujących nowe funkcje.

W podsumowaniu należy podkreślić nowatorskie rozwiązania zastosowane w projekcie i implementacji (niebanalne algorytmy, nowe technologie, itp.).

Bibliografia

[1] Constraint programming. Web pages: https://www.math.unipd.it/~frossi/chapter4.pdf [ostatni dostęp: 07.11.2022].

[2] Fast planning through planning graph analysis. Web pages: https://www.cs.cmu.edu/~avrim/Papers/graphplan.pdf/ [ostatni dostęp: 21.10.2022].

[3] A modern treatment of the 15 puzzle, aaron f. archer. Web pages: https://www.cs.cmu.edu/afs/cs/academic/class/15859-f01/www/notes/15-puzzle.pdf [ostatni dostęp: 07.11.2022].

[4] The parallel search bench zram and its applications. Web pages: http://www.iro.umontreal.ca/~gendron/Pisa/References/BB/Brungger99.pdf [ostatni dostęp: 07.11.2022].

[5] Solving the 15-puzzle. Web pages: https://cseweb.ucsd.edu/~ccalabro/essays/15˙puzzle.pdf [ostatni dostęp: 07.11.2022].

[6] Strips: A new approach to the application of theorem proving to problem solving. Web pages: http://ai.stanford.edu/~nilsson/OnlinePubs-Nils/PublishedPapers/strips.pdf [ostatni dostęp: 07.11.2022].

[7] N. P. Russell Stuart. *Artificial Intelligence A Modern Approach (4th Edition)*. Pearson, 2020.

Załącznik A

Zawartość płyty CD

W tym rozdziale należy krótko omówić zawartość dołączonej płyty CD.